НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)

“___” червня 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки **6.050102 «Комп'ютерна інженерія»**

на тему: Графічна бібліотека з використанням паралельних обчислень на GPGPU

Виконав: студент IV курсу, групи КВ-53
(шифр групи)

Кузьмін В’ячеслав Вікторович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник старший викладач Дробязко І.П. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М.

(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент д.т.н., професор Сімоненко В.П. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ІАЛЦ.045492.003 ТП	Відомість технічного проекту	1	
3	А4	ІАЛЦ.045492.004 ПЗ	Пояснювальна записка	54	
4	А4	ІАЛЦ.045492.005 Д1	Графічна бібліотека з використанням паралельних обчислень на GPGPU. Схема структурна.	1	
5	А4	ІАЛЦ.045492.006 Д2	Графічний конвеєр. Схема структурна	1	
6	А4	ІАЛЦ.045492.007 Д3	Генерація лінії на екрані. Схема алгоритму.	1	
7	А4	ІАЛЦ.045492.008 Д4	Візуалізація тривимірного зображення. Схема алгоритму.	1	

				ДП ХХХХ 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.	Кузьмін В.В			Відомість дипломного проекту	Лист	Листів
Керівн.	Дробязко І.П.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. СПіСКС Гр. КВ-53	
Н/контр.	Клятченко Я.М.					
Зав.каф.	Тарасенко В.П.					

Пояснювальна записка
до дипломного проекту

на тему: Графічна бібліотека з використанням паралельних обчислень на GPGPU
_____ _
_____ _

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050102 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)
«___» червня 2019 р.

ЗАВДАННЯ
на дипломний проект студента
Кузьміна В’ячеслава Вікторовича
(прізвище, ім’я, по батькові)

1. Тема проекту “Графічна бібліотека з використанням паралельних обчислень на GPGPU”, керівник проекту старший викладач Дробязко Ірина Павлівна, затверджені наказом по університету №1330-С від 22.05.2019.

2. Термін подання студентом проекту «21» травня 2019 р.

3. Вихідні дані проекту: див. Технічне завдання

4. Зміст пояснювальної записки:

- Аналіз проблем сучасної ком’ютерної графіки;
- Аналіз побудови тривимірного зображення;
- Опис алгоритмів побудови тривимірного зображення;
- Тестування та порівняння швидкодії бібліотеки;

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо)

- Графічна бібліотека з використанням паралельних обчислень на GPGPU. Схема структурна

- Графічний конвеєр. Схема структурна
- Генерація лінії на екрані. Схема алгоритму
- Візуалізація тривимірного зображення. Схема алгоритму
- Презентація на тему проекту

6. Консультанти розділів проекту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормконтроль	Клятченко Я.М.		

7. Дата видачі завдання _____.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	15.11.2018	
2.	Розроблення та узгодження технічного завдання	30.11.2018	
3.	Аналіз існуючих рішень	05.02.2019	
4.	Підготовка матеріалів першого розділу дипломного проекту	05.03.2019	
5.	Підготовка матеріалів другого розділу дипломного проекту	27.03.2019	
6.	Підготовка матеріалів третього розділу дипломного проекту	15.04.2019	
7.	Підготовка графічної частини дипломного проекту	10.05.2019	
8.	Оформлення документації дипломного проекту	14.05.2019	
9.	Попередній огляд матеріалів диплому на кафедрі	25.05.2019	

Студент _____ _____
(підпис) (ініціали, прізвище)

Керівник проекту _____ _____
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломного проекту.

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (54 с., 36 рис. 1 табл., 5 додатки).

Об’єкт розробки – створення графічної бібліотеки, яка дозволяє візуалізовувати тривимірні об’єкти в реальному часі та відслідковувати кожний крок процесу.

Графічна бібліотека постачає fine-grained API який дозволяє будувати зображення на моніторі комп’ютера в реальному часі. Low-level API дозволяє будувати графічні примітиви, high-level API – тривимірні сцени. За допомогою додаткових функцій можливо здійснювати лінійні перетворення: масштабування, паралельний зсув, поворот, тощо. Для оптимальної швидкодії використовується гетерогенна архітектура (використовується CUDA SDK, фірми NVIDIA), тобто всі графічні обчислення виконуються на GPU (Graphics Processing Unit) з використанням великої кількості паралельних потоків. CPU (Central Processing Unit) керує розподілом пам’яті та обчислювальними потоками.

У даному дипломному проекті розроблено: гетерогенну архітектуру графічно бібліотеки, fine-grained API, алгоритм візуалізації тривимірних зображень з Wavefront OBJ файлу, реалізація фрагментного та вершинного шейдерів, алгоритм побудови тіней, читання текстур коліру, нормалей та освітленності.

Ключові слова: комп’ютерна графіка, CUDA, алгоритм растеризації, графічна бібліотека, паралельні обчислення.

ABSTRACT

The qualifying work includes an explanatory note (54 p., 36 figures, 1 tables, 5 pts).

The object of development is the creation of a graphical library that allows you to visualize 3D objects in real time and track each step of the process.

The graphical library implements a fine-grained API that allows you to build images on a computer monitor in real time. The low-level API allows you to build graphic primitives, high-level APIs - three-dimensional scenes. With the help of additional functions, it is possible to make linear transformations: zoom, parallel shift, rotation, etc. For optimal performance, a heterogeneous architecture is used (used by CUDA SDK, NVIDIA), all graphical computations are executed on a GPU (Graphics Processing Unit) using a large number of parallel streams. The CPU (Central Processing Unit) controls the allocation of memory and computing flows.

The following structures and algorithms are developed in this project: a heterogeneous graphical library architecture, a fine-grained API, an algorithm for visualizing three-dimensional images from a Wavefront OBJ file, implementing fragment and vertex shaders, a shadow algorithm, color texturing, normality, and illumination.

Key words: computer graphics, CUDA, rasterization algorithm, graphical library, parallel computing.

АННОТАЦИЯ

Квалификационная работа включает пояснительную записку (54 с. 36 рис. 1 табл. 5 приложений).

Объект разработки - создание графической библиотеки, которая позволяет визуализировать трехмерные объекты в реальном времени и отслеживать каждый шаг процесса.

Графическая библиотека поставляет fine-grained API который позволяет строить изображение на мониторе компьютера в реальном времени. Low-level API позволяет строить графические примитивы, high-level API - трехмерные сцены. С помощью дополнительных функций можно осуществлять линейные преобразования: масштабирование, параллельный сдвиг, поворот, и тому подобное. Для оптимальной производительности используется гетерогенная архитектура (с помощью CUDA SDK, фирмы NVIDIA), то есть все графические вычисления выполняются на GPU (Graphics Processing Unit) с использованием большого количества параллельных потоков. CPU (Central Processing Unit) управляет распределением памяти и вычислительными потоками.

В данном дипломном проекте разработаны: гетерогенная архитектура графической библиотеки, fine-grained API, алгоритм визуализации трехмерных изображений с Wavefront OBJ файла, реализация фрагментного и вершинные шейдеры, алгоритм построения теней, чтения текстур цвета, нормалей и освещенности.

Ключевые слова: компьютерная графика, CUDA, алгоритм растрирования, графическая библиотека, параллельные вычисления.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість	№	Примітки
	А4	ІАЛЦ.045492.002	Графічна бібліотека з	4		
			використанням			
			паралельних обчислень			
			на GPGPU			
			Технічне завдання			
	А	ІАЛЦ.045492.003	Графічна бібліотека з	1		
			використанням			
			паралельних обчислень			
			на GPGPU			
			Відомість проекту			
	А	ІАЛЦ.045492.004	Графічна бібліотека з	5		
			використанням			
			паралельних обчислень			
			на GPGPU			
			Відомість проекту			
	А	ІАЛЦ.045492.005	Графічна бібліотека з	1		
			використанням			
			паралельних обчислень			
			на GPGPU			
			Схема структурна			

					ІАЛЦ.045492.002 ОА
	А	№ докум.	Під	Д	
Розроб.		Кузьмін В.В.			Графічна бібліотека з використанням паралельних обчислень на GPGPU Опис альбому
Перевір.		Дробязко			
Н. контр.		Клятченко			
Затвер.		Тарасенко			

Літ.	Ар	Аркуш
	1	2

КПІ ім. Ігоря Сікорського ФПМ КВ-52

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість	№	Примітки
	А	ІАЛЦ.045492.006	Графічний конвеєр	1		
			Схема структурна			
	А	ІАЛЦ.045492.007	Генерація лінії на екрані	1		
			Схема структурна			
	А	ІАЛЦ.045492.008	Візуалізація	1		
			зображення			
			Схема алгоритму			
		Диск CD-ROM	Матеріали дипломного	1		
			проекту.			

					ІАЛЦ.045492.002 ОА	Арк.
	А	№ докум.	Підп	Д		2

Зміст

					ІАЛЦ.045492.002 ТЗ		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розробив		Кузьмін В.В.			Графічна бібліотека з використанням паралельних обчислень на GPGPU Технічне завдання	Літ.	Аркуш 1 / Аркушів 4
Перевірив		Дробязко І.П.					
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського ФПМ КВ-53	
Затвердив		Тарасенко В.П.					

НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Найменування роботи – «Графічна бібліотека з використанням паралельних обчислень на GPGPU».

Область застосування: інформаційні технології, комп’ютерна графіка.

ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на виконання першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп’ютерних систем Національного технічного університету України «Київський Політехнічний Інститут ім. Ігоря Сікорського».

МЕТА РОЗРОБКИ

Метою даного проекту є розробка графічної бібліотеки, яка надає змогу візуалізувати тривимірні графічні об’єкти формату Wavefront OBJ file з використанням карт текстур та обчислень на графічному процесорі. Бібліотека дає змогу зрозуміти прикладним програмістам процеси які відбуваються на нижніх рівнях абстракції, алгоритми які апаратно реалізовані в сучасних графічних прискорювачах, роботу спеціалізованих графічних бібліотек таких як OpenGL та DirectX.

ДЖЕРЕЛА РОЗРОБКИ

Джерелами інформації для розроблення є технічна література, публікації у періодичних виданнях та спеціалізовані Інтернет ресурси.

ТЕХНІЧНІ ВИМОГИ

Вимоги до додатку, що розробляється

Графічна бібліотека повинна реалізовувати такий функціонал:

- Читання OBJ моделі з диску в пам’ять комп’ютера.
- Візуалізація моделі на екран комп’ютера згідно конфігурацією користувача.
- Виконувати примітивні графічні перетворення, такі як: поворот, масштабування, переміщення відносно системи координат.
- Побудова графічних примітивів: точка, лінія, багатокутник, багатокутник з заливкою.
- Надання додаткової інформації: кількість графічних прискорювачів та тип, інформація про CUDA SDK, кількість полігонів у графічного об’єкта, FPS (Frames Per Second).

Вимоги до апаратного забезпечення

- Процесор: Intel Core i3 7^{th} Gen
- Оперативна пам’ять: 256 Мб.
- Вільна пам’ять на диску: 100 Мб.
- Графічний прискорювач: NVIDIA з підтримкою технології CUDA.

Вимоги до програмного забезпечення

- Операційна система MS Windows 7, MS Windows 8, MS Windows 10.
- Наявність встановленої бібліотеки SDL 2.0.
- Наявність графічних драйверів.
- CUDA SDK.

ЕТАПИ ПРОЕКТУВАННЯ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.11.2018
2.	Розроблення та узгодження технічного завдання	30.11.2018
3.	Аналіз існуючих рішень	05.02.2019
4.	Підготовка матеріалів першого розділу дипломного проекту	05.03.2019
5.	Підготовка матеріалів другого розділу дипломного проекту	27.03.2019
6.	Підготовка матеріалів третього розділу дипломного проекту	15.04.2019
7.	Підготовка графічної частини дипломного проекту	10.05.2019
8.	Оформлення документації дипломного проекту	14.05.2019
9.	Попередній огляд матеріалів диплому на кафедрі	25.05.2019

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість	№	Примітки
	А4	ІАЛЦ.045492.004 ПЗ	Пояснювальна записка	54		
			Графічна документація			
	А4	ІАЛЦ.045492.005 Д1	Графічна бібліотека з використанням паралельних обчислень на GPGPU	1		
			Схема структурна			
	А4	ІАЛЦ.045492.006 Д2	Графічний конвеєр	1		
			Схема структурна			
	А4	ІАЛЦ.045492.007 Д3	Генерація лінії на екрані	1		
			Схема алгоритму			
	А4	ІАЛЦ.045492.008 Д4	Візуалізація тривимірного зображення	1		
			Схема алгоритму			
		CD-диск	Матеріали дипломного	1		

					ІАЛЦ.045492.003 ТП
А		№ докум.	Підпи	Д	
Розробив	Кузьмін В.В.			Графічна бібліотека з використанням паралельних обчислень на GPGPU. Відомість технічного проекту	Літ. / Арк 1 / Аркуші 1
Перевірив	Дробязко І.П.				
Н. контроль	Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-53
Затвердив	Тарасенко В.П.				

ЗМІСТ

ДОДАТКИ

Додаток 1. Копії графічних матеріалів

- ІАЛЦ.045492.005 Д1. Графічна бібліотека з використанням паралельних обчислень на GPGPU. Схема структурна
- ІАЛЦ.045492.006 Д2. Графічний конвеєр. Схема структурна
- ІАЛЦ.045492.007 Д3. Генерація лінії на екрані. Схема алгоритму
- ІАЛЦ.045492.008 Д4. Візуалізація тривимірного зображення. Схема алгоритму

Додаток 2. Формат Wavefront OBJ файлу

Додаток 3. Презентація

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

ОЗП – оперативний запам'ятовуючий пристрій

API - Application Programming Interface (прикладний інтерфейс користувача)

GPU – Graphics Processing Unit (графічний процесор)

CPU - Central processing unit (центральний процесор)

GPGPU – General-purpose computing on graphics processing units (загальні обчислення на графічному процесорі)

CUDA – Compute Unified Device Architecture (уніфікована архітектура обчислювальних пристроїв)

SDK – Software Development Kit (комплект для разработки программного обеспечения)

ISA – Industry Standard Architecture (промислова стандартна архітектура)

SIMD – Single Instruction, Multiple Data (один потік команд і множинний потік даних)

SIMT – Single Instruction, Multiple Threads (один потік команд та множинний потік виконання)

DLL – Dynamic-link library (динамічно приєднувана бібліотека)

RLE – Run-length encoding (кодування довжини виконання)

T & L – transformation and light (трансформування та освітлення)

FPS – frames per second (кількість кадрів за секунду)

ВСТУП

В сучасному світі інформаційних технологій комп’ютерна графіка знайшла застосування практично у всіх сферах суспільства. Неможливо уявити наукові дослідження без візуалізації результатів за допомогою спеціалізованого графічного програмного забезпечення, кіноіндустрію без використання об’єктів, повністю створених на комп’ютері, книжки без намальованих у графічних редакторах ілюстраціях тощо. Комп’ютерна графіка ініціювала створення нової популярної індустрії - комп’ютерні ігри, яка невпинно розвивається і набуває все більшого масштабу.

Галузь комп’ютерної графіки є достатньо молодою і на даний час швидко розвивається. Створюються нові, більш ефективні та гнучкі технології і програмні засоби, удосконалюються вже відомі. Комп’ютерна графіка – комплексна наука, яка складається з багатьох підрозділів, в основі яких лежать складні математичні моделі та алгоритми. Все це неможливо охопити одній людині. Галузь комп’ютерної графіки потребує все більшої кількості високо професійних спеціалістів, які змогли б задовольнити все більш зростаючий попит.

До основних напрямків застосування комп’ютерної графіки можна віднести: графічний інтерфейс користувача, анімація, векторна графіка, тривимірне моделювання, симуляція фізичних поверхонь, комп’ютерний зір, кіноіндустрія, комп’ютерні ігри, наукова візуалізація тощо. Вони потребують кваліфікованих фахівців різної спрямованості та області знань.

Для чіткого та повного розуміння предмету недостатньо просто прочитати технічну документацію, важливо отримати практичний досвід, провести експерименти та дослідити кожний крок візуалізації тривимірних об’єктів самостійно. На сьогоднішній час це неможливо – сучасні графічні бібліотеки приховують від користувача всі обчислення, що в більшості випадків добре, адже програміст витрачає значно менше часу на розробку інфраструктури ПЗ та концентрується на написанні функціональної частини. Але для розуміння це крок назад, недосвідчений програміст може неефективно використовувати бібліотеку, що призводить до нераціональних витрат ресурсів ПК та користувача.

Таким чином, актуальним є розробка бібліотеки, яка, на відміну від наявних, дозволяє досліджувати кожний крок її виконання. Бібліотека дозволить зробити процес візуалізації зображення прозорим та зрозумілим для програміста і створювати ефективні програмні засоби для роботи з комп’ютерною графікою.

1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМУ

1.1 Аналіз проблем сучасної комп'ютерної графіки

Комп'ютерна графіка, як і багато інших молодих наук, має багато невирішених питань. Головною задачею комп'ютерної графіки є отримання зображення, максимально схожого на зображення реального світу, за оптимальний для користувача час.

Однією із проблем є розробка високопродуктивного та потужного апаратного забезпечення для прискорення візуалізації тривимірної та двовимірної графіки. Архітектурні рішення та характеристики (зокрема пропускна здатність) графічних прискорювачів вдосконалюються з плином часу, і за короткий проміжок часу дозволили створити складні та швидкодіючі системи, здатні обчислювати мільйони пікселів на екрані щосекунди.

Вдосконалювати комп'ютерну графіку завжди шляхом покращення апаратних засобів неможливо. Необхідно використовувати складну математичну теорію та комплекси програмного забезпечення, які вирішують специфічні для завдань графіки проблеми. На сьогодні існує багато програмних засобів, які вирішують як специфічні проблеми комп'ютерної графіки, так і базові, на яких базується решта засобів.

Існує багато невирішених проблем в сфері комп'ютерної графіки:

- Ефективний прорахунок тіней тривимірної сцени для реалістичного зображення займає занадто багато часу, неможливо точно прорахувати всі тіні у режимі реального часу.

- Візуалізація складних тривимірних сцен з високою деталізацією неможлива в реальному часі, оскільки тривимірний об'єкт описується набором вершин, а для деталізованих тривимірних об'єктів кількість вершин може досягати десятки мільйонів. Сучасні програмні та апаратні засоби не мають змоги візуалізувати такі об'єкти в реальному часі. Для вирішення цієї проблеми вже є можливе рішення – воксельна графіка (voxel graphics), де тривимірний об'єкт

описується набором вокселів – еквівалентом пікселів у тривимірному просторі.

- Реалістична симуляція волосся, трави та подібних складних об’єктів, які вимагають значної кількості ресурсів для прорахунку; на сьогодні ця проблема не вирішена, і використовується компроміс між реалістичністю та швидкістю візуалізації кадру.

- Ефективна візуалізація таких фізичних явищ, як сніг, попіл, іскри та інше. На сьогодні їх візуалізація є одними із найбільш ресурсномістких частин графічного конвеєру, його рішення балансує між кількістю частинок та унікальністю їх поведінки.

Для розробки та підтримки існуючих програмних рішень потрібно все більше професійних розробників та винахідників, які будуть вдосконалювати засоби комп’ютерної графіки. На сьогодні процес навчання для роботи з комп’ютерною графікою є доволі складним і вимагає значних ресурсів часу та теоретичної підготовки програміста. Сучасні графічні бібліотеки приховують багато базових механізмів та алгоритмів від прикладного програміста, що значно ускладнює процес навчання, особливо для початківців.

1.2 Аналіз існуючих рішень

Існує багато варіантів API (Application Programming Interface) та графічних бібліотек для роботи з комп’ютерною графікою, кожна з яких має свої переваги та недоліки, але жодна з них не дозволяє програмісту повністю прослідкувати процес візуалізації зображення на моніторі комп’ютера.

DirectX – розроблено для простого і ефективного вирішення задач, пов’язаних з ігровим відео-програмуванням для операційної системи Microsoft Windows.

Бібліотека являє собою набір COM об’єктів, які поділяються на: DirectX Graphics (виведення графіки на екран), DirectInput (обробка даних з периферійних пристроїв, таких як клавіатура, мишка, ігровий контролер тощо), DirectPlay (засоби для мережевої

комутації), DirectSound (засоби для роботи з простим звуком), DirectMusic (засоби відтворення музики) тощо.

OpenGL – крос-платформна бібліотека для написання програм, які працюють з 3D та 2D зображеннями, містить набір функцій для генерації деталізованих тривимірних зображень, широко використовується у графічних інтерфейсах користувача, відеоіграх, візуалізації наукових даних та системах автоматизованого проектування тощо. OpenGL є специфікацією, на основі якої виробники апаратного забезпечення створюють власні реалізації – бібліотеки функцій, що відповідають заявленим критеріям.

Сьогодні існує декілька версій цієї бібліотеки:

- OpenGL 2.0 –до ядра OpenGL додано GLSL (мову програмування рейдерів).
- OpenGL 3.0 – підтримує нову версію GLSL 1.30, масиви текстур, 32-бітовий буфер глибини з плаваючою крапкою, нові режими стискування текстур та ін.
- OpenGL 4.0 – додає нові етапи для обробки зображення за допомогою шейдерів, забезпечує 64-бітову точність операцій з плаваючою точкою, збільшення продуктивності.

Vulkan – крос-платформне API для 3D графіки, що створене як заміна застарілому OpenGL при вирішенні існуючих проблем. Основною метою створення цієї бібліотеки було більш раціональне використання можливостей GPU (Graphics Proccesing Unit, графічний процесор) і CPU (Central processing unit, центральний процесор). Бібліотека впроваджує прямий контроль над роботою GPU і зменшує навантаження на CPU.

В основі лежить раціональне використання можливостей графічного прискорювача та центрального процесора в порівнянні з DirectX і OpenGL. Vulkan, на відміну від OpenGL, впроваджує шейдери у проміжному бінарному форматі під назвою SPIR-V, що дає змогу виконувати компіляцію шейдерів вже на етапі розробки, а не виконання програми. Це уможливлює не тільки зменшення навантаження на центральний процесор, а й створення шейдерів на інших мовах програмування.

Також бібліотека краще підтримує багатоядерні процесори, що забезпечує значний приріст швидкодії, але ускладнює розробку програмного забезпечення.

Mantle – API низького рівня, що розроблене під архітектуру AMD Radeon, як спроба заміни DirectX та OpenGL. Забезпечує більшу швидкодію за рахунок концентрації на одній апаратній платформі.

Всі апаратні можливості, які реалізовані у графічному прискорювачі, можливо викликати через API бібліотеки, що дозволяє ефективно використовувати апаратні засоби. Бібліотека дозволяє реалізувати прямий доступ до пам'яті графічного прискорювача.

На сьогодні компанія AMD повністю зупинила розробку та підтримку графічної бібліотеки.

1.3 Обґрунтування теми дипломного проекту та постановка задачі

Усі представлені вище графічні бібліотеки широко застосовуються при розробці програмного забезпечення, але жодна з них не дозволяє повністю відслідковувати процес візуалізації зображення. Це значно ускладнює вивчення кожної з бібліотек і оптимальну розробку алгоритмів та програм, які базуються на обраній бібліотеці.

Таким чином, актуальною є розробка бібліотеки, яка, на відміну від розглянутих вище, дозволяє відслідковувати та досліджувати кожний крок графічного конвеєра. Після експериментальних досліджень з використанням функцій графічної бібліотеки програміст краще засвоїть основи сучасної візуалізації тривимірних зображень, що дозволить використовувати здобуті знання для розробки ефективних проектів на базі професійних графічних бібліотек.

Дана графічна бібліотека повинна забезпечувати наступні основні функціональні можливості та вимоги:

- Візуалізація повинна здійснюватися в реальному часі;
- Підтримка операцій переміщення, повороту та масштабування;
- Завантаження будь-якого тривимірного об'єкту із файлової системи;
- Читання зображення із файлової системи;
- Відображення часу візуалізації одного кадру;
- Бути простою у використанні та інтегруванні в будь-який програмний проект;
- Простий процес відлагоджування та покрокового виконання;

- Створення та підключення власних шейдерів;
- Завдання розмірів вікна;
- Якість нового отриманого або сформованого зображення повинна бути не гіршою ніж у аналогів;
- Підтримка, за замовчуванням, буфера глибини;
- Зручне налаштування будь-яких параметрів візуалізації.

2. ТЕХНОЛОГІЧНІ ТА ПРОГРАМНІ ЗАСОБИ РОЗРОБКИ

2.1 Аналіз способів побудови тривимірного зображення

Рендерінг (англ. rendering) – це генерування зображення тривимірного об’єкта на моніторі комп’ютера за допомогою спеціальних алгоритмів. Модель – це опис певної тривимірної сцени за допомогою чітко визначеної структури даних. Слово рендерінг в Україні вживають для позначення процесу візуалізації, а рендер — для позначення готового зображення. Тобто існують синоніми: комп'ютерний рендерінг — комп'ютерна візуалізація, рендер — візуалізований об'єкт.

Комп’ютерна візуалізація – одна з базових частин комп’ютерної графіки, яка використовується практично у всіх областях. Для різних задач створюють спеціалізоване, в залежності від вимог задачі, програмне забезпечення для отримання найкращої якості вихідного зображення.

Існують два типи рендерінгу: пре-рендерінг, який дозволяє візуалізовувати сцени, максимально наближені до реального світу, (є досить затратним по ресурсам та не може працювати в реальному часті) і рендерінг у режимі реального часу, який застосовується у комп'ютерних іграх.

Фотореалістична візуалізація (пре-рендерінг) застосовується для створення складних, максимально наближених до реальності сцен, в яких відбувається складний прорахунок тіней та оптичних властивостей світла (приклад на рис. 2.1). Пре-рендерінг найчастіше застосовується в кіноіндустрії, комп’ютерній графіці (графічних редакторах). Така візуалізація не може відбуватись в реальному часі, оскільки потребує значного часу для створення зображення (для прорахунку одного кадру може знадобитись 3-5 хвилин, а для прорахунку всього фільму – декілька місяців), та візуалізується за допомогою рендер ферм – великої кількості потужних ЕОМ, спеціально створених для роботи з комп’ютерною графікою [4].

Рисунок 2.1 – Приклад фотореалістичного рендеру

Нефотореалістична візуалізація (в реальному часі) застосовується для створення максимально наближених до реального світу сцен, проте здатних працювати в режимі реального часу (приклад на рис. 2.2.). Застосовується в комп'ютерних іграх, спеціальних симуляціях та наукових дослідженнях.

Рисунок 2.2 – Приклад нефотореалістичного рендеру

Відомі наступні методи візуалізації тривимірних сцен:

- Растерізація (англ. rasterization) – візуалізація проводиться проекцією всіх об'єктів сцени на двовимірну площину без розрахунків перспективи.
- Метод кидання променів (англ. ray casting) – сцена візуалізується з певної точки (точки зору спостерігача), з цієї ж точки пускаються промені на об'єкти сцени і за

допомогою променів визначають колір пікселя на двовимірній площині (приклад на рис. 2.3). При цьому поширення променю припиняється при досяганні об’єкта сцени або її фону. Таким чином отримується ефект перспективи без додаткових обчислень.

Рисунок 2.3 – Зображення за допомогою кидання променів

- Трасування променів (англ. ray tracing) – подібне до методу кидання променів, але промінь при попаданні на об’єкт не припиняє свого поширення, а розбивається на 3 промені, кожний з яких коригує колір пікселя на двовимірному екрані: відбитий, тіньовий та заломлений. Результатом роботи цього методу є високо реалістичні зображення, проте він є досить витратним за часом. Приклад зображення, отриманого трасуванням променів, показано на рис. 2.4 [1].

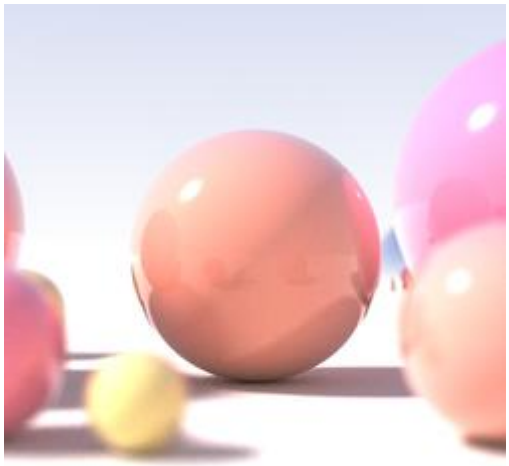


Рисунок 2.4 – Зображення за допомогою методу трасування променів

- Трасування шляху (англ. path tracing) – методика рендерінга в комп'ютерній графіці, здатна симулювати фізичні характеристики світу настільки близько до реальності, наскільки це можливо (приклад зображення показано на рис. 2.5). Трасування шляху є узагальненням традиційного трасування променів, алгоритм якого передбачає трасування прямих в напрямку від віртуальної камери через простір; пряма «відскакує» від предметів до тих пір, поки повністю не розсіється або поглинеться будь-яким матеріалом. Якість зображень, отриманих за допомогою методів трасування шляху, як правило, краща, ніж якість зображень, отриманих іншими методами рендерінга, проте трасування шляху вимагає більших обчислювальних витрат.

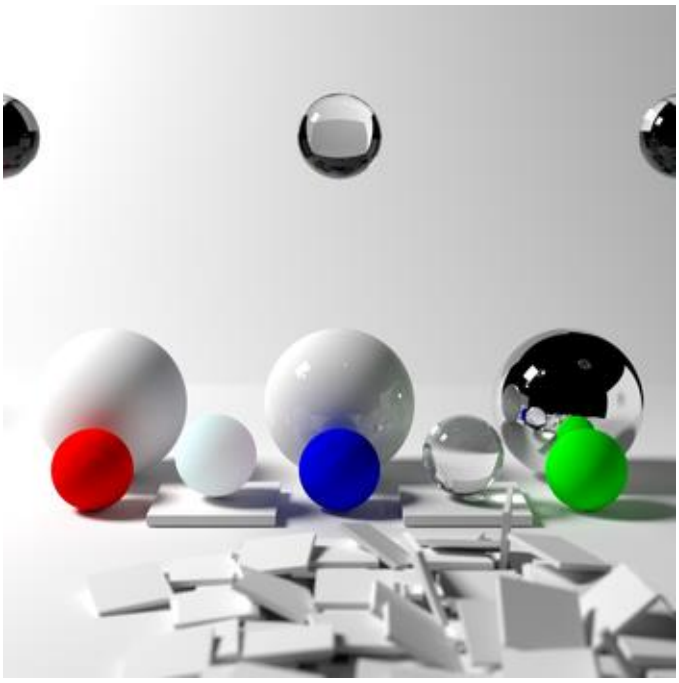


Рисунок 2.5 – Зображення за допомогою трасування шляху

2.2 Обґрунтування вибору технологій розроблення

Візуалізація тривимірних об’єктів – складний та витратний процес. Для візуалізації існують спеціальні апаратні рішення – графічні прискорювачі, які беруть на себе всі складні розрахунки. Основні операції у графічному прискорювачі реалізовано апаратно, тому простежити весь процес неможливо.

Даний проект реалізується для навчальних цілей, тому важливо забезпечити можливість відстеження кожного кроку виконання візуалізації. Всі операції в графічній бібліотеці повинні реалізовуватись програмно. Реалізація за допомогою можливостей тільки центрального процесору не є оптимальною. Для досягнення задовільної швидкодії потрібно використовувати гетерогенну архітектуру, де всі складні обчислення будуть виконуватись за допомогою допоміжного процесору, тобто виконуватись на GPGPU (General-Purpose Computing on Graphics Processing Units, (загальні обчислення на графічному процесорі).

Гетерогенні обчислювальні системи – системи, які використовують для обчислень більш ніж один процесор з різною архітектурою. В якості обчислювальних пристроїв можуть бути, наприклад, графічний процесор або будь-який інший співпроцесор. Гетерогенна обчислювальна система включає в себе процесори з різними наборами інструкцій (ISA) [9].

За допомогою гетерогенної архітектури, яку дозволяє створювати бібліотека CUDA, програміст зможе самостійно відстежувати та модифікувати кожний крок виконання алгоритму побудови тривимірного зображення.

2.3. Опис технологій та мов програмування

Даний проект використовує мову C та C++ для максимального контролю над апаратними можливостями ПК без зайвих рівнів абстракції. C++ дозволяє ефективно виконувати маніпуляції з пам’яттю. Оскільки код мови компілюється в код асемблеру, на відміну від Java або Python, які є інтерпретованими мовами програмування, C++ виконується на порядок швидше.

Для реалізації гетерогенних обчислень використовується CUDA SDK. CUDA (Compute Unified Device Architecture) – спеціально розроблена архітектура для паралельних обчислень на графічному прискорювачі, яка виконує всі складні розрахунки в багатоядерному середовищі графічних процесорів фірми nVidia.

CUDA SDK (Compute Unified Device Architecture Software Development Kit) дозволяє включати в текст програм виклик спеціальних підпрограм, написаних на спеціальному діалекті мови C\C++, які виконуються на графічних процесорах. Бібліотека надає

можливість програмісту використовувати та організовувати доступ до графічного прискорювача та відеопам’яті.

Графічна бібліотека оптимізована спеціально для графічних процесорів nVidia і тому є більш швидкодіючою, ніж аналоги. Мінусом використання бібліотеки є неможливість переносу програми на інші графічні прискорювачі, наприклад AMD, а також не всі графічні прискорювачі nVidia підтримують дану архітектуру, їх обчислювальна здатність значно залежить від класу графічного прискорювача.

CUDA – це бібліотека для гетерогенних обчислень, тобто дозволяє використовувати процесор загального призначення та процесор спеціального призначення, який контролюється ЦП. Тобто, гетерогенна обчислювальна платформа містить процесори з різними наборами ISA (Industry Standard Architecture, промислова стандартна архітектура) команд (центральний процесор та графічний прискорювач). Центральний процесор використовує SIMD (Single Instruction, Multiple Data, одиночний потік команд, множинний потік даних) блоки для векторних обчислень. Архітектура CUDA використовує SIMT (Single Instruction, Multiple Threads, одна інструкція та декілька потоків) для скалярної обробки потоків на графічному присорювачі. Процесор загального призначення викликає спеціальну функцію LaunchKernel, для якої задає потрібну кількість потоків для виконання, після чого процес виконання переходить на графічний прискорювач, який виконує їх відповідно заданій конфігурації. Загальна схема виконання представлена на рис. 2.6.

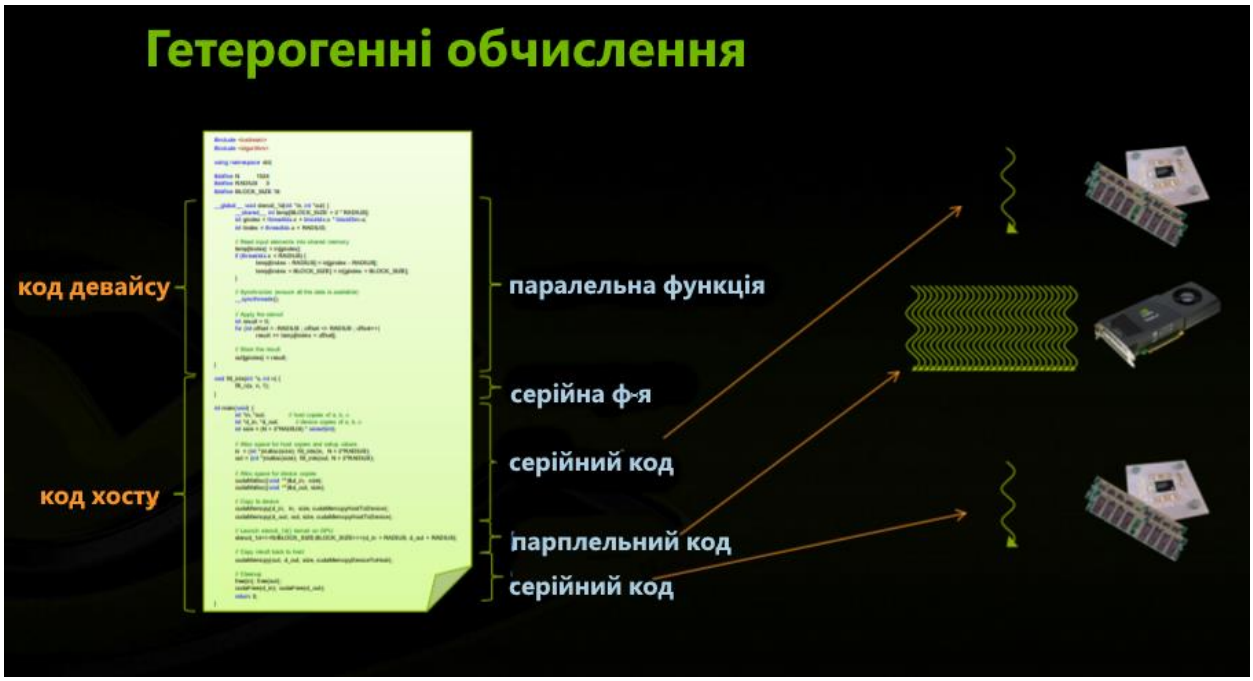


Рисунок 2.6 – Виконання інструкцій з використанням CUDA

Оскільки сучасні ОС не дозволяють копіювати згенероване зображення безпосередньо в пам’ять графічного прискорювача, використовується бібліотека SDL 2.0.

SDL – крос-платформна мультимедійна бібліотека з відкритим вихідним кодом, яка реалізує єдиний програмний інтерфейс до графічної підсистеми. Бібліотека розроблена мовою програмування С, включає в себе не тільки графічний інтерфейс, але й роботу із звуком та пристроями введення. Ця бібліотека є однією з найпростіших та потужніших, що і вплинуло на її використання в даному проекті. Бібліотека SDL надає такі можливості, як швидке виведення 2D-графіки, відстеження натиснутих клавіш клавіатури, програвання звуку, виведення 3D-графіки через OpenGL і багато інших корисних операцій у крос-платформному коді. У даному проекті вона використовується як додатковий рівень абстракції, який дозволяє формувати зображення на екрані комп’ютера.

3. ОПИС РОЗРОБКИ

3.1. Архітектура системи

Розроблена бібліотека являє собою DLL файл (Dynamic Link Library, динамічно приєднувана бібліотека). DLL – це бібліотека, яка містить код і дані, що можуть використовуватися кількома програмами одночасно [2]. Це дає змогу доволі просто та зручно підключати бібліотеку до будь-якого проекту, написаного навіть іншою мовою програмування. DLL приховує від користувача всі деталі реалізації, що дозволяє створювати зручне та зрозуміле API. Також бібліотека може включатись в код користувача безпосередньо (код тільки мовами C та C++). Це може бути корисно, якщо програма повинна повністю розміщуватися у виконуваному файлі (EXE file).

Бібліотека складається з набору модулів (див. рис. 3.1), які взаємодіють між собою. Кожний модуль має окреме призначення і міститься у своєму просторі імен, що корисно не тільки для чіткої структури, а й для запобігання конфліктів у коді користувача. Основний простір імен бібліотеки – gl, який містить в собі всі модулі бібліотеки:

- Cuda_geometry – реалізує основні типи та алгоритми для математичних обчислень (клас матриці, вектору та дії над ними);
- Camera – реалізує поворот камери та перспективне перетворення;
- RenderOnGPU – реалізує функції для завантаження та початку візуалізації тривимірного об’єкту;
- ModelBuffer – реалізує операції читання та доступу до тривимірної моделі;
- TextureBuffer – реалізує операції читання та доступу до TGA зображення;
- Screen – реалізує операції виведення візуалізованого зображення на монітор комп’ютера;
- Drawing – реалізує базові операції з двовимірною графікою;

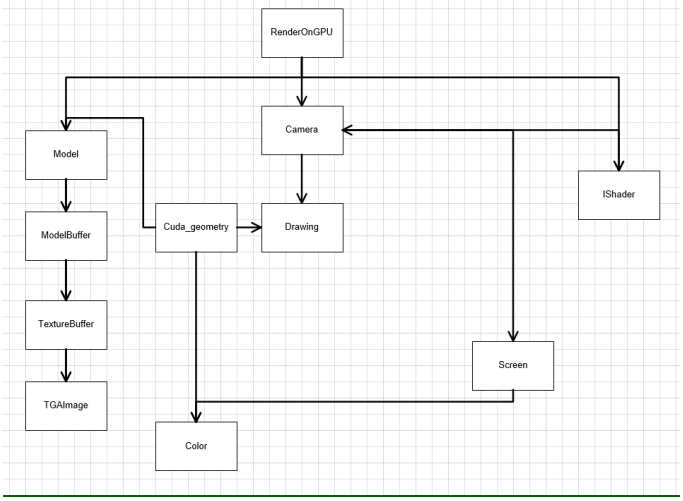


Рисунок 3.1 – Схема взаємодії модулів бібліотеки

3.2. Опис модулів бібліотеки

Модуль Model та ModelBuffer. Читання та зберігання опису тривимірного об'єкта із файлу

Базова задача зберігання 3D об'єктів – представлення інформацію про об'єкт у вигляді простого тексту або бінарних даних. Тривимірні об'єкти містять геометрію, матеріали (текстури, матеріали, тощо) і анімацію, яку важливо ефективно зберігати та читати.

За читання та зберігання файлу опису тривимірного об'єкту в оперативному запам'ятовуючому пристрої (ОЗП) комп'ютера відповідає модуль Model. Для маніпуляцій над об'єктом в пам'яті графічного прискорювача використовується модуль ModelBuffer, призначення якого полягає у копіюванні об'єкта в пам'ять графічного прискорювача та виконанні операцій доступу до моделі за допомогою коду, що виконується на графічному прискорювачі.

Формат зберігання не зобов'язаний підтримувати все перелічене вище, наприклад STL зберігає тільки геометрію і ігнорує всі інші параметри, формат COLLADA підтримує все. На сьогодні існують сотні форматів для зберігання 3D об'єктів. Кожна програма для моделювання використовує свій, оптимізований для себе формат, однак існують і формати з відкритим вихідним кодом, які є досить поширеними та популярними. Найпоширенішими форматами: OBJ, STL, FBX, COLLADA, 3DS, IGES, STEP, VRML\X3D.

Розроблена графічна бібліотека читає файли тільки формату OBJ. Він досить простий та поширений, задає тільки геометрію об’єкта: координати кожної вершини, текстурні координати, нормалі та грані. Вершини многокутників за замовчуванням задаються проти годинникової стрілки, що робить явне завдання нормалей необов’язковим. Приклад структури формату OBJ наведено у Додатку 2.

Модуль TGAImage та ImageBuffer. Читання та зберігання карт текстур

Текстура – растрове зображення, яке накладається на тривимірний об’єкт для реалістичного відтворення поверхні. Вона дозволяє деталізувати об’єкт, оскільки деталізація за допомогою полігонів (трикутників) є занадто повільною і не може застосовуватись в реальному часі. Наприклад, деталізація обличчя людини за допомогою текстур є досить простою, на відміну від деталізації збільшенням полігонів. Зображення без текстур наведено на рис. 3.2.

Текстури бувають шовні та безшовні (патерни або візерунки). Якість поверхні визначається текселями – кількістю пікселів на мінімальну одиницю текстури. Роздільність текстури та формат зберігання у файловій системі сильно впливають на якість результуючого зображення об’єкта на екрані.

Для відтворення певного візуального зображення на об’єкті використовують карти текстур. Процес накладання карт текстур схожий на застосування візерунчастого паперу на коробці для подарунків. Кожній вершині об’єкту присвоюються координати текстури (UV coordinates). В процесі візуалізації текстура інтерполюється згідно цих координат, і в результаті ми отримуємо текстурований об’єкт. Карти текстур накладаються одна на одну (мультиплексування) для комбінування ефектів.

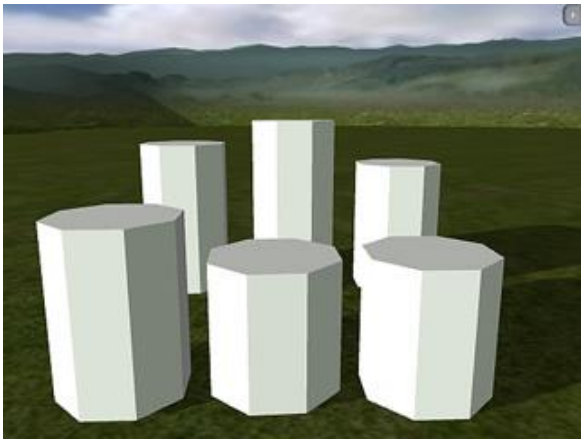


Рисунок 3.2 – Тривимірні об'єкти без текстур

Існує декілька типів карт текстур:

- Diffuse map (укр. карта кольору) – найбільш поширений тип текстур, за допомогою якої задається колір кожної точки об’єкта. Для подальшої коректної роботи інших карт текстур, текстура не повинна мати ніякого спрямованого світла на поверхню. Приклад зображено на рис 3.2.

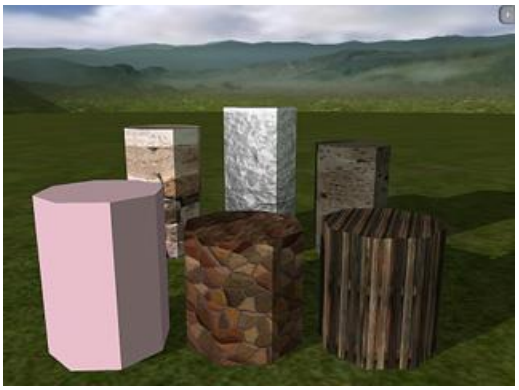


Рисунок 3.2 – Тривимірні об'єкти з картою кольору

- Specular map (укр. карта відбиття) – містить пікселі в чорно-білій гамі; чим світліший піксель, тим більша здатність матеріалу відбивати світло і тим яскравіші на ньому відблиски від світла. Приклад карти відбиття показано на рис 3.3. Відповідно, чим піксель темніший, тим більш матовим стає матеріал і втрачає свою властивість відбивати світло. Для таких матеріалів, як керамічна плитка і полірований метал, можуть використовуватися світліші тони, для тканин і дерева – темні.

Рисунок 3.3 – Тривимірний об'єкт із картою відбиття

- Bump map (укр. карта рельєфу) – містить пікселі в градаціях сірого кольору. Приклад можна побачити на рис 3.4. При рендерінгу сірі області карти відображатимуться як звичайно; чорні будуть затемнюватись, а білі освітлюватись – така ілюзія світла створює ефект рельєфу поверхні.

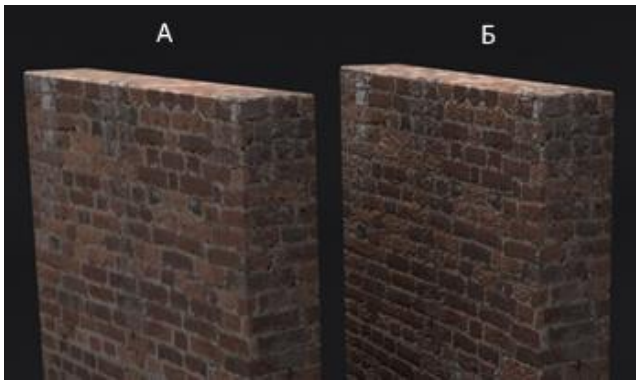


Рисунок 3.4 – (А) об’єкт без карти рельєфу і (Б) з картою рельєфу

- Normal map (укр. карта нормалей) – дозволяє змінювати нормаль пікселя, опираючись на колір пікселя з карти нормалей (RGB), на основі якого обчислюється нормаль. Приклад використання показано на рис 3.5. Даний метод є точнішим ніж bump map за рахунок використання трьох каналів, на відміну від одного у карті рельєфу.

Рисунок 3.5 – Об'єкт з використанням та без карт нормалей

- Opacity map (укр. карта непрозорості) – містить чорно-білі пікселі, де повністю чорний піксель є прозорим, білий – повністю непрозорим. Градація задає різний ступінь прозорості. Приклад використання наведено на рис 3.6.

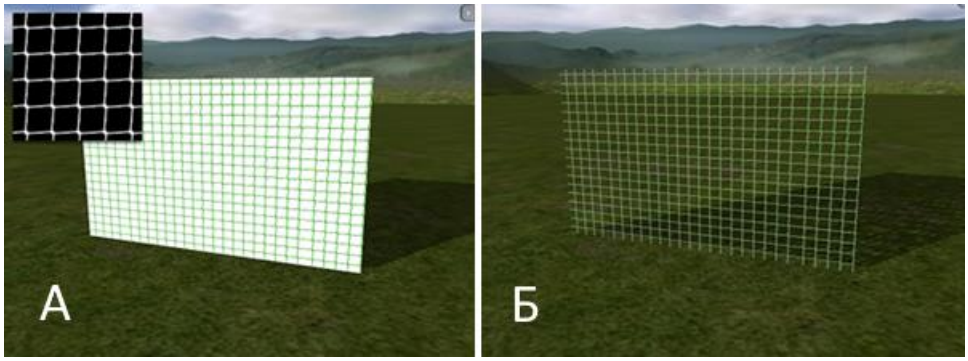


Рисунок 3.6 – (А) Об'єкт без карти прозорості, (Б) із картою прозорості

Модуль читання зображень

Модуль читання зображень потрібний для зчитування та декодування файлів текстур в пам’ять комп’ютера для подальшого їх використання в програмі. Дана бібліотека підтримує читання зображень формату TGA – це растровий графічний формат, створений компанією Truevision. Формат підтримує глибину кольору 1-32 біти на піксель, альфа-канали, стискування в формат RLE (Run-length encoding) [3].

Файл зображення складається з чотирьох секцій, кожна з яких містить свої поля (останні три поля є необов’язковими, і, в залежності від версії формату, можуть бути відсутні):

1. Заголовок файлу.

2. Зображення або мапа кольорів.
3. Зона розробника.
4. Зона розширення.

Модуль складається з таких субмодулів: TGAImage, TGAColor, ImageBuffer. Розглянемо їх детальніше:

TGAImage – основний інструмент для роботи з TGA зображенням, дозволяє читати та зберігати зображення з файлової системи, виконувати із зображенням маніпуляції, такі як: масштабування, відображення по горизонталі та вертикалі. За допомогою цього інструменту можна задати певний піксель новим кольором, прочитати колір пікселя тощо.

TGAColor – допоміжна структура, яка описує колір пікселя і дозволяє маніпулювати ним.

ImageBuffer – об’єкт для зберігання зображення в пам’яті графічного прискорювача, дозволяє читати колір кожного пікселя за допомогою інструкцій графічного прискорювача.

Модуль читання файлів опису тривимірних об’єктів

Модуль містить два класи Model та ModelBuffer:

- Model зберігається в оперативній пам’яті комп’ютера та дозволяє читати файли формату OBJ з файлової системи, автоматично читає всі карти текстур, які містить об’єкт за допомогою модуля читання зображень. Основна задача класу – декодування зображення в ОЗП комп’ютера для зручної роботи з тривимірним об’єктом.
- ModelBuffer дозволяє копіювати об’єкт з оперативної пам’яті комп’ютера у пам’ять графічного прискорювача та виконувати всі маніпуляції з об’єктом, використовуючи обчислювальні можливості відеокарти.

Модуль для роботи з екраном комп’ютера

Об’єкт Screen дозволяє ефективно заповнювати екран згенерованим графічною бібліотекою зображенням. Він дозволяє створювати вікно заданої висоти та ширини, задавати колір будь-якого пікселя, очищати екран. Оскільки ОС не дозволяє напряму

записувати в буфер графічного прискорювача, бібліотека дозволяє ефективну зміну буфера кадру з незначними втратами швидкодії.

Об’єкти h_Color та d_Color виконують однакові за можливостями функції для роботи з кольором пікселя, такі як завдання кольору у зручному для користувача вигляді: RGBA форматі, кожний канал складається з одного байту, тобто кожний колір можна задати в межах від 0 до 255. Тобто, загалом за допомогою об’єкту можна представити 16 581 375 унікальних кольорів (з альфа каналом – 4 228 250 625 унікальних кольорів). Префікс на початку назви означає, що колір зберігається або в ОЗП комп’ютера, або у пам’яті графічного прискорювача: d_ – device (графічний прискорювач), h_ – host (центральний процесор), та відповідно всі обчислення виконуються на різних апаратних пристроях.

Модуль графічних обчислень

Модуль Cuda_geometry є одним із основних для всієї бібліотеки. Він реалізує всі складні математичні обчислення для роботи з тривимірною графікою. В ньому описуються та реалізовуються базові примітиви лінійної алгебри – вектори та матриці, які можуть бути довільної довжини. Це досягається за допомогою вбудованого механізму шаблонів мови C++ та їх підтримкою архітектурою CUDA.

Модуль повністю створено для виконання на графічному прискорювачі та оптимізовано під архітектуру CUDA. В ньому реалізовані такі операції як: початкове заповнення векторів та матриці заданими значеннями, транспонування матриці, основні арифметичні операції над векторами та матрицями (додавання, віднімання, множення, ділення), нормалізація векторів, обчислення детермінантів матриці та визначено зручні псевдоніми типів для користувачів.

Всі функції модуля виконуються в процесі візуалізації зображення багато разів, дуже важливо досягти максимально можливої швидкості виконання функцій, адже різниця навіть в одну асемблерну інструкцію може відчутно збільшити швидкість візуалізації.

Модуль Drawing

Растеризація (англ. rasterisation) – процес перетворення векторного зображення (представленого у вигляді геометричного опису) у растрове зображення (представленого як колекція пікселів – точок на екрані монітора) [4].

За растеризацію зображення відповідає модуль Drawing, який виконує всі операції для виведення пікселей на екран. Він реалізований на розширеній для архітектури CUDA мові програмування C++.

Модуль включає наступні функції для тривимірної візуалізації:

- setPixel – задає певний колір заданого пікселя на екрані.
- Line – малює пряму між двома точками із заданим кольором за допомогою алгоритму Брезентхема.
- Triangle – малює трикутник із заливкою заданим кольором.
- triangleZBuf – малює трикутник із заливкою заданим кольором, враховується буфер глибини.
- tirangleWithText – малює трикутник із заливкою заданим кольором, враховується буфер глибини та накладає текстуру кольору.
- Triangle_s – малює трикутник із заливкою заданим кольором, використовує технологію шейдерів.

А також функції для двовимірного малювання:

- SetRectangle – малює прямокутник за заданими координатами із гранями заданого кольору.
- SetCircle – малює коло із центром в заданій точці та з заданим радіусом.
- SetPolygon – малює довільну послідовність з’єднаних прямих, що задаються користувачем.
- SetImage – малює задане користувачем зображення на екрані.

Шейдери

Шейдер – комп’ютерна програма, що виконується графічними прискорювачами. Програми, працюючі з тривимірною графікою і відео, використовують шейдери для

визначення параметрів геометричних об’єктів або зображень, для зміни зображення (для створення ефектів зсуву, відображення, заломлення, затемнення з урахуванням заданих параметрів поглинання і розсіювання світла, для накладання текстур на геометричні об’єкти тощо) [5].

Раніше шейдери поділяли на три типи, в залежності від задачі, яку вони виконують: вершинний (англ. vertex shader), геометричний (англ. geometry shader) та піксельний або фрагментний (англ. pixel shader, fragment shader). Сьогодні усі графічні прискорювачі підтримують уніфіковану шейдерну архітектуру – шейдери виконуються на одному із багатьох графічних субпроцесорів і поділяються тільки на рівні абстракцій, на відміну від різних апаратних засобів. Розглянемо детальніше застосування кожного типу шейдера:

- Вершинний шейдер оперує даними, зв’язаними з вершинами багатогранників, наприклад, з координатами вершини в просторі, з текстурними координатами, з кольором вершини, з вектором нормалі тощо. Вершинний шейдер може використовуватись для видового або перспективного перетворення вершин, для генерації текстурних координат, для розрахунку освітлення тощо.
- Геометричний шейдер, на відміну від вершинного, здатний обробляти не тільки одну вершину, але й цілий примітив. Примітивом може бути відрізок (дві вершини) і трикутник (три вершини), а, при наявності інформації про суміжні вершини для трикутного примітива, може бути оброблено до шести вершин. Геометричний шейдер здатний генерувати примітиви «on the fly» (на етапі виконання програми), не навантажуючи при цьому центральний процесор.
- Піксельний шейдер працює з фрагментами растрового зображення і з текстурами – обробляє дані, зв’язані з пікселями (наприклад колір, глибина, текстурні координати). Піксельний шейдер використовується на останній стадії графічного конвеєра для формування фрагмента зображення.

Шейдери пишуться на спеціалізованих мовах програмування, в даному проекті – за допомогою CUDA. Шейдер розроблюється спеціальною розширеною для CUDA мовою C.

В розробленій бібліотеці інтерфейс шейдера описується в модулі IShader. Кожна реалізація власного шейдера повинна наслідуватись від IShader. Піксельний шейдер виконується для кожного пікселя зображення, тому код функції повинен виконуватися з максимально можливою швидкодією.

3.3.Опис використаних алгоритмів

Видалення невидимих поверхонь. Алгоритм z-буфера

Задача видалення невидимих поверхонь є однією із найважчих в комп'ютерній графіці. Алгоритм повинен аналізувати всю видиму для спостерігача сцену і, в залежності від позиції об'єкта сцени відносно спостерігача, вирішити, чи приховується об'єкт іншим, який знаходиться ближче до спостерігача, і, якщо він приховується, видалити об'єкт із процесу візуалізації. Таким чином, сцена не тільки набуває реалістичності, а й відбувається значний приріст швидкодії.

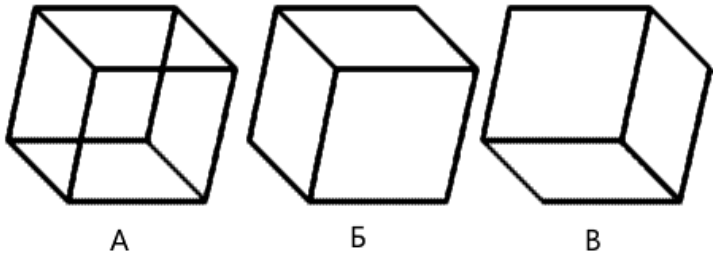


Рисунок 3.7 – Необхідність видалення невидимих поверхонь

На рисунку 3.7 (А) представлено типову каркасну модель куба, його можна інтерпретувати двояко: як вигляд куба згори зліва чи знизу праворуч. Видалення тих ліній та поверхонь, які невидимі з заданої точки зору, дозволяє позбавитись від неоднозначності. Результати показано на рис. 3.7 (Б) і (В).

Ідеального алгоритму вирішення задачі видалення поверхонь немає. Для комп'ютерної анімації або кінематографу, наприклад, потрібні алгоритми для генерації складних реалістичних зображень, в яких присутні тіні, детальна фактура об'єктів та підтримка прозорості, враховувати рефлексію кольору та складні оптичні ефекти. Такі алгоритми візуалізації повільні і можуть витрачати на побудову одного кадру до 2-3 хвилин, а на побудову великого фрагменту відео до декількох місяців. Побудова

відблисків, деталізація об’єктів, прорахунок тіней не входить до завдання видалення невидимих ліній. Ці задачі є частиною процесу візуалізації, часто вони вбудовані в алгоритм видалення невидимих поверхонь. Це залежить від задачі. Важливо знайти компроміс між якістю результуючого зображення та швидкістю його побудови. Зі створенням все більш швидких алгоритмів відповідно можна будувати все більш деталізовані та реалістичні зображення.

Алгоритми видалення невидимих ліній чи поверхонь базуються на сортуванні. Для кожної сцени відбувається сортування об’єктів за відстанню об’єкта від точки зору глядача. Основна ідея сортування за відстанню полягає в тому, що чим далі розташовано об'єкт від точки спостереження – тим більша ймовірність, що він буде закритий одним з об'єктів, ближчих до точки спостереження. Після визначення відстаней або пріоритетів за глибиною, залишається провести сортування по горизонталі і по вертикалі, щоб з'ясувати, чи буде даний об'єкт дійсно закрито об'єктом, розташованим ближче до точки спостереження.

Від ефективності реалізації алгоритму сортування залежить загальна швидкодія виконання алгоритму видалення невидимих поверхонь. Для підвищення ефективності сортування використовується також когерентність сцени, тобто тенденція незмінності характеристик сцени [11].

Алгоритми видалення невидимих ліній або поверхонь можна класифікувати за способом вибору системи координат або простору, в якому вони працюють. Виділяють три класи алгоритмів видалення невидимих ліній або поверхонь:

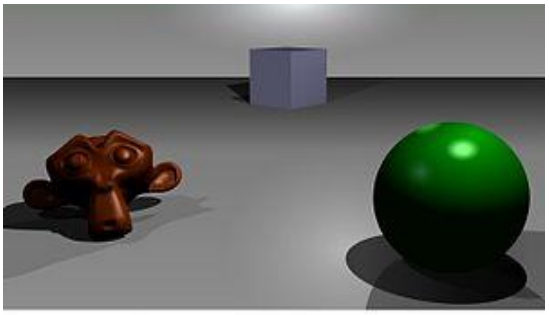
- Алгоритми, що працюють в об’єктному просторі.
- Алгоритми що працюють в просторі зображення (екрану).
- Алгоритми, що формують список пріоритетів.

Алгоритми, що працюють в об'єктному просторі, мають справу з фізичною системою координат, в якій описані ці об'єкти. При цьому вихідні результати є дуже точними та обмеженими лише точністю обчислень. Отримані зображення можна вільно збільшувати в

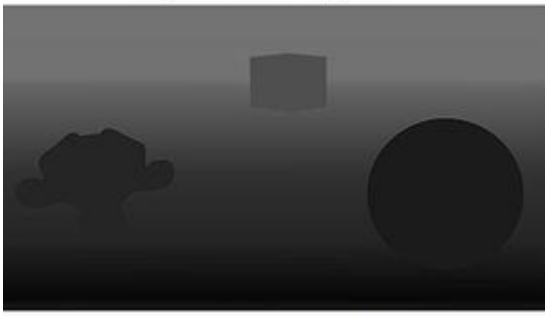
багато разів. Алгоритми, що працюють в об'єктному просторі, особливо корисні в тих випадках, коли необхідна висока точність.

Алгоритми, що працюють в просторі зображення, починають працювати з системою координат того екрану, на якому об'єкти зображуються. При цьому точність обчислень обмежена роздільною здатністю екрану. Збільшені у багато разів зображення не будуть відповідати вихідній сцені. Наприклад, можуть не збігатись кінці відрізків.

В даній бібліотеці використовується алгоритм Z-буферізації. При візуалізації тривимірного об'єкту його глибина аналізується на осі Z-координат і зберігається у Z-буфері. Сам буфер являє собою двомірний масив координат із глибиною для кожного пікселя. Коли інший об'єкт сцени потрібно відобразити у цьому ж пікселі, порівнюються дві глибини. Якщо об'єкт знаходиться ближче до спостерігача, піксель перекривається. Обрана глибина зберігається в Z-буфері і замінює попередню. Зрештою, Z-буфер дозволяє правильно відтворювати звичне для нас сприйняття глибини: ближчий до спостерігача об'єкт перекриває наступні, що розташовуються за ним. Приклад використання та приклад вигляду Z-буфера, збереженого до спеціального зображення в тестових цілях, можна побачити на рисунку 3.8.



Проста 3D сцена



Її відображення у Z-буфері

Рисунок 3.8 – Сцена із використанням Z-буфера та приклад Z-буфера

Побудова перспективного зображення

На фотографіях, картинах або екрані зображення здаються природними і правильними, тому що в них зберігається перспектива, яка характерна для зору людини. Її властивість така, що більш віддалені предмети зображуються в менших масштабах, а паралельні прямі, у загальному випадку, непаралельними. Як результат цього, геометрія зображення виявляється досить складною, і по готовому зображенню складно визначити розмір тих чи інших частин об'єкта, що наглядно показано на рис. 3.9.

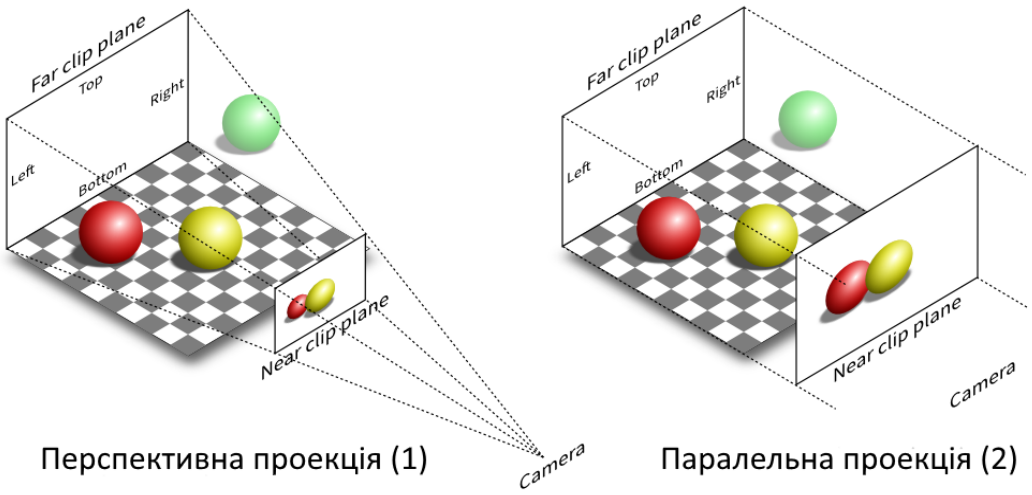


Рисунок 3.9 – Параллельна (2) та перспективна (1) проекції

Для побудови реалістичного зображення потрібно відтворити перспективу, максимально приближену до реального світу. Перспектива надає відчуття глибини тривимірного зображення.

Розглянемо проблему візуалізації тривимірних зображень на двовимірну площину. Для цього необхідно визначити необхідні математичні моделі. Потрібно враховувати багато факторів, які впливають на візуальне сприйняття реальних образів людським оком. Спосіб переходу від тривимірних об'єктів до їх зображень на площині називатимемо проекцією [6].

Проекції перетворять точки, задані в системі координат розмірністю n, в точки системи координат розмірністю меншою ніж n. У даному випадку, точки тривимірного простору перетворюються в точки двовимірного простору. Проекції будуються за допомогою променів або проекторів, які виходять з точки, що називається центром проекції. Проектори проходять через площину, яка називається проекційною або картинною площиною, і потім проходять через кожну точку тривимірного об'єкту, утворюючи тим самим проекцію. Тип проектування на пласку, а не викривлену поверхню, де в якості проекторів використовуються прямі. а не викривлені лінії, називається плоскою геометричної проекцією. Пласкі геометричні проекції діляться на два види: центральні або перспективні (рис. 3.9 (1)) та паралельні або ортогональні (рис. 3.9 (2)). Якщо центр проекції перебуває на кінцевій відстані від проекційної площини, тоді це центральна

проекція. Якщо центр проекції віддалений до нескінченності, тоді це паралельна проекція [10].

В розробленій графічній бібліотеці виконується проектування на площину z.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & r & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ rz+1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ rz+1 \end{bmatrix} \rightarrow \begin{bmatrix} \frac{x}{rz+1} \\ \frac{y}{rz+1} \\ \frac{z}{rz+1} \end{bmatrix} \quad (3.1)$$

Головна ідея проекції: чим глибше знаходиться предмет, тим більше стає значення z-координати і знаменника rz + 1 (3.1), отже, тим дрібніше виглядає предмет на площині проекції.

Лінійні операції над тривимірною сценою

Проаналізуємо правосторонню декартову систему координат з трьома вимірами. Якщо поворот проти годинникової стрілки на 90° буде переводити одну піввісь в іншу, вважатимемо поворот позитивним. На основі цих правил побудуємо таблицю 3.1, яку можна використовувати як для лівих, так і для правих систем координат [8].

Таблиця 3.1 – Додатні повороти за віссю обертання

Якщо вісь обертання	Додатнім буде напрямок повороту
X	Від y до z
Y	Від z до x
Z	Від x до y

У тривимірному просторі точка описується за допомогою вектора (x, y, z). Для виконання трансформацій точок у тривимірному просторі розглянемо узагальнену матрицю перетворення:

$$\begin{bmatrix} a & b & c & p \\ d & e & f & q \\ h & i & j & r \\ l & m & n & s \end{bmatrix} \quad (3.2)$$

Матриця, що зображена на рис. 3.2, може розглядатись як 4 окремі частини:

$$\begin{bmatrix} 3\times3 & 3\times1 \\ 1\times3 & 1\times1 \end{bmatrix} \quad (3.3)$$

- Матриця 3x3 виконує лінійне перетворення, наприклад зміни масштабу, зсуву і обертання.
- Матриця 1x3 виконує перенесення.
- Матриця 3x1 виконує перспективне перетворення.
- Скалярний елемент 1x1 використовується для загального масштабування.

Розглянемо вплив матриці 4x4 (вираз 3.3) на однорідний вектор [x, y, z, 1]:

- Тривимірне перенесення є простим розширенням двовимірного:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Dx & Dy & Dz & 1 \end{bmatrix} \quad (3.4)$$

Тобто $[x,y,z,1]*T(Dx, Dy, Dz)=[x+Dx,y+Dy,z+Dz,1]$.

- Тривимірна зміна масштабу. Розглянемо часткову зміну масштабу. Вона реалізується у такий спосіб:

$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.5)$$

Тобто $[x,y,z,1]*S(Sx, Sy, Sz)=[Sx*x, Sy*y, Sz*z,1]$.

Загальна зміна масштабу виходить за рахунок 4-го діагонального елемента:

$$[x\ y\ z\ 1] * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & S \end{bmatrix} = [x\ y\ z\ S] = [x^*\ y^*\ z^*\ \ 1] = [\frac{X}{S}, \frac{Y}{S}, \frac{Z}{S}, 1]. \quad (3.6)$$

Такий самий результат можна отримати при однакових коефіцієнтах часткових змін масштабів. У цьому випадку, матриця перетворення наступна:

$$S=\begin{bmatrix} \frac{1}{s_x} & 0 & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 & 0 \\ 0 & 0 & \frac{1}{s_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

- Тривимірний зсув. Недіагональні елементи матриці 3x3 здійснюють зрушення в трьох вимірах, тобто:

$$[x\ y\ z\ 1] * \begin{bmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ h & i & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x+yd+hz,\ bx+y+iz,\ cx+fy+z,\ 1]. \tag{3.8}$$

- Тривимірний поворот.

  У тривимірному просторі поворот навколо осі Z описується матрицею:

$$\begin{bmatrix} \cos(\Theta) & \sin(\Theta) & 0 & 0 \\ -\sin(\Theta) & \cos(\Theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

  Матриця повороту навколо осі X має вигляд:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Theta) & \sin(\Theta) & 0 \\ 0 & -\sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

  Матриця повороту навколо осі Y має вигляд:

$$\begin{bmatrix} \cos(\Theta) & 0 & -\sin(\Theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\Theta) & 0 & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

В процесі візуалізації кожного кадру виконується суперпозиція всіх матриць для отримання кінцевого результату всіх видів перетворень (показаних на матрицях 3.4, 3.5, 3.8, 3.9).[10]

Графічний конвеєр

Для ефективної візуалізації у всіх сучасних графічних бібліотеках використовують графічний конвеєр – апаратно-програмний комплекс візуалізації тривимірної графіки, в якому етапи виконуються послідовно і паралельно. Він дозволяє прискорити виконання візуалізації за рахунок одночасного виконання різних етапів обробки. На сьогодні, більшість виробників графічних прискорювачів реалізують графічний конвеєр апаратно, оскільки такий підхід дозволяє значно підвищити швидкодію системи. У загальному вигляді, сучасний прискорювач 3D-графіки повинен мати блок геометричних перетворень, розрахунку освітлення (геометричний процесор), блок механізму установки примітивів, блок обробки текстур і блок обробки буфера кадру. Геометричний процесор обробляє примітиви і будує проекцію тривимірної сцени. Блок розрахунку освітлення формує дані про параметри освітлення для вершин примітивів. Типовий графічний конвеєр представлено на рис. 3.10. В даному проекті для спрощення сприйняття графічна бібліотека використовує спрощену модель графічного конвеєра.

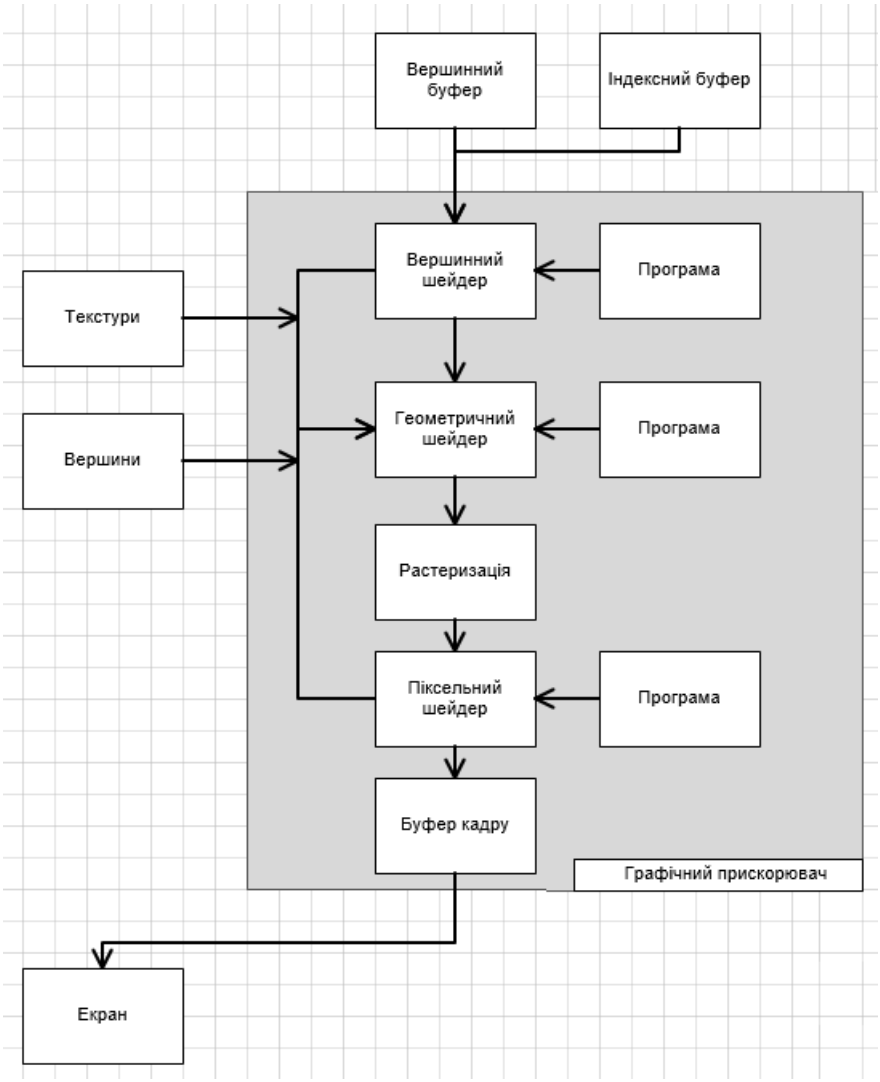


Рисунок 3.10 – Приклад графічного конвеєра

Виділяють шість кроків роботи графічного конвеєра [8]:

1. Визначається набір і стан об'єктів, які необхідні для виведення зображення на поточний момент. Відбувається вибірка присутніх в сцені в даний момент об'єктів, тобто видимих з поточної позиції камери. Вони додаються до динамічного списку, готового до обробки. Решта об'єктів, що знаходяться за межами огляду, або повністю заховані за іншими об'єктами, відкидаються і не використовуються для подальшої обробки. Всі дані, отримані на цьому етапі, повністю обробляються центральним процесором. Результат пересилається у графічний прискорювач.

2. Відбувається збірка примітивів з отриманих на попередньому кроці даних: координат і індексів вершин. Масив з індексами зчитується, збираючи дані з масиву координат в трикутники, за вказаною кількістю байтів на кожну вершину (4 байти – вершина типу float, 8 байтів – вершина типу double). Також кожній вершині присвоюється її колір. Результати відправляються в блок T & L (трансформацій і освітлення).
3. У блоці T & L до вершин трикутників застосовуються різні ефекти освітленості і перетворень. Процес перетворення блоків T & L сучасних графічних прискорювачів можна змінювати за допомогою вершинних шейдерів – спеціальних програм, що вбудовуються в основний код. На разі персональний комп'ютер має повністю програмований графічний процесор, що надало технологічний ривок у використанні тривимірної графіки. По завершенні операцій розрахунку освітленості і трансформації параметри вершин нормалізуються.
4. Сьогодні графічні процесори підтримують ще додаткові дії, наприклад теселяцію – процес додавання нових трикутників у полігональну сітку для збільшення деталізації об’єкту, тобто поділ початкових трикутників на дрібніші. Геометричні шейдери можуть не тільки теселювати полігони, але також в реальному часі створювати тривимірні примітиви, ґрунтуючись на отриманих даних, у тому числі спеціальних текстур, що на практиці, за рахунок багатоядерності графічного процесора, дозволяє виводити високо деталізовану графіку набагато швидше, ніж за допомогою центрального процесора.
5. Графічний прискорювач практично завжди має багато паралельних текстурних конвеєрів. На кожен видимий піксель відбувається накладання кольору в залежності від заданих текстур і параметрів змішування, які програмуються за допомогою піксельних шейдерів. Також можлива робота з тими текстурами, які не слугують для відображення, а використовуються для модифікації кінцевого кольору, (наприклад, карти висот, карти освітлення). Також тут використовується інформація про належність пікселя до певного трикутнику і його координати на двовимірній площині полігону.

6. На кінцевій стадії роботи графічного конвеєра до отриманого зображення застосовують пост-ефекти. До них відносяться різні фільтрації і операції по усуненню дефектів. Дані виводяться в кадровий буфер, і зображення з'являється на дисплеї.

Всі перераховані вище дії можуть виконуватись в іншому порядку, бути виконані неодноразово, в залежності від заданого рішення і технологічних можливостей комп'ютера. Частина дій може виконуватися програмно, а частина – апаратно, в залежності від типу відеоприскорювача. Найбільш сучасні прискорювачі мають графічний процесор, який спроможний апаратно прораховувати етапи трансформації, розрахунку освітлення і накладення текстури.

3.4 Інтерфейс користувача

Для тестування бібліотеки розроблено спеціальну програму для переглядання тривимірних зображень формату Wavefront OBJ. Вона використовує всі можливості бібліотеки.

Головне вікно, що представлено на рис. 3.11, складається з трьох частин:

1. Заголовок, який показує кількість кадрів за секунду в реальному часі та кнопки згортання і закриття вікна.
2. Панель налаштувань може довільно переміщуватись в області візуалізації та містить всі налаштування додатку.
3. Область візуалізації, в якій відбувається показ тривимірної моделі.

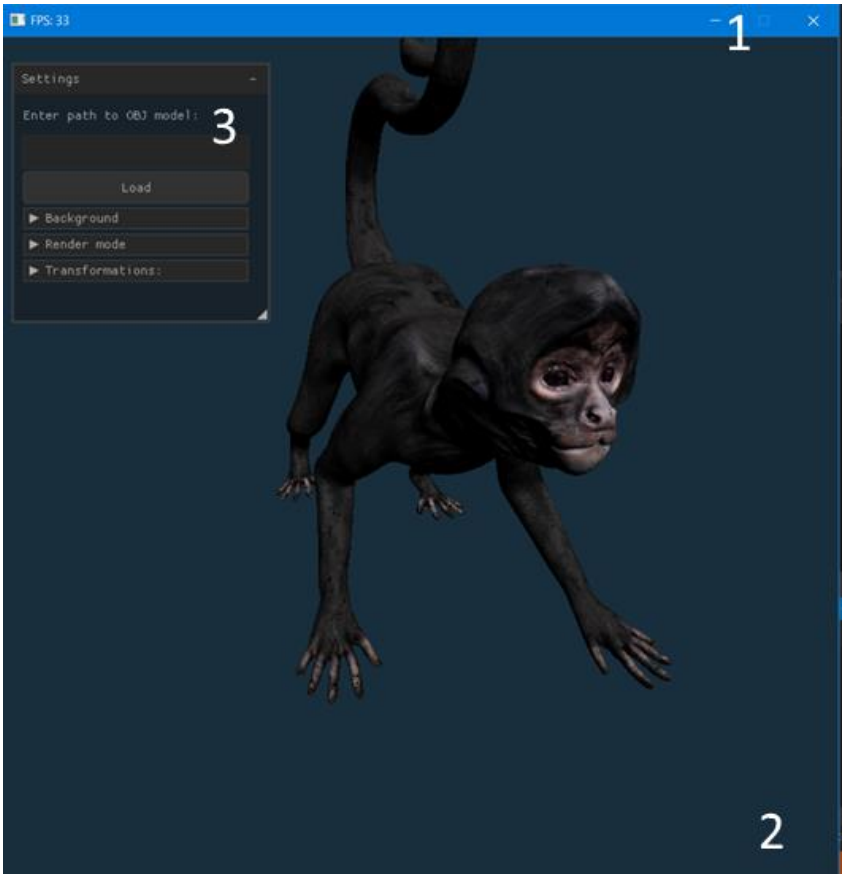


Рисунок 3.11 – Головне вікно програми

Вікно налаштувань можна вільно переміщувати або згортати (рис. 3.12). Для більшої зручності перегляду області візуалізації вікно налаштувань є напівпрозорим.

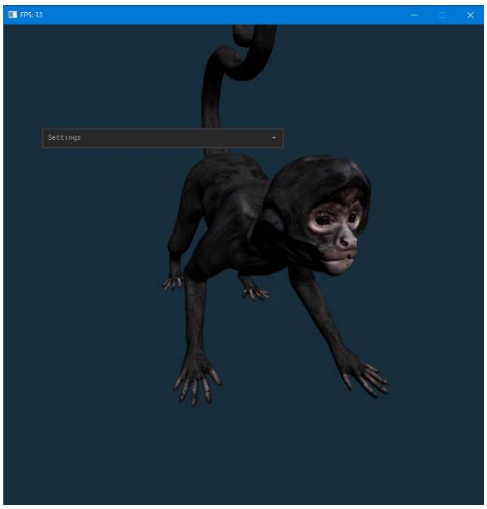


Рисунок 3.12 – Приклад переміщення та згортання панелі налаштувань

Розглянемо детальніше вікно налаштувань. Вікно має засоби для відкриття будь-якого Wavefront OBJ файлу та набір підменю, кожне з яких відповідає за налаштування спільної групи параметрів. Розглянемо кожне підменю окремо (перелік виконується зверху-вниз):

Для відкриття довільної тривимірної моделі існує поле “Enter path to OBJ model”, в яке можна вписати шлях до файлу та натиснути кнопку “Load”, після цього у вікні візуалізації з’явиться прочитаний за заданим шляхом тривимірний об’єкт. При цьому відкриття нового тривимірного об’єкта може зайняти деякий час, в залежності від розміру файлу та текстур, які задані в ньому. Результат виконання показано на рис. 3.13.

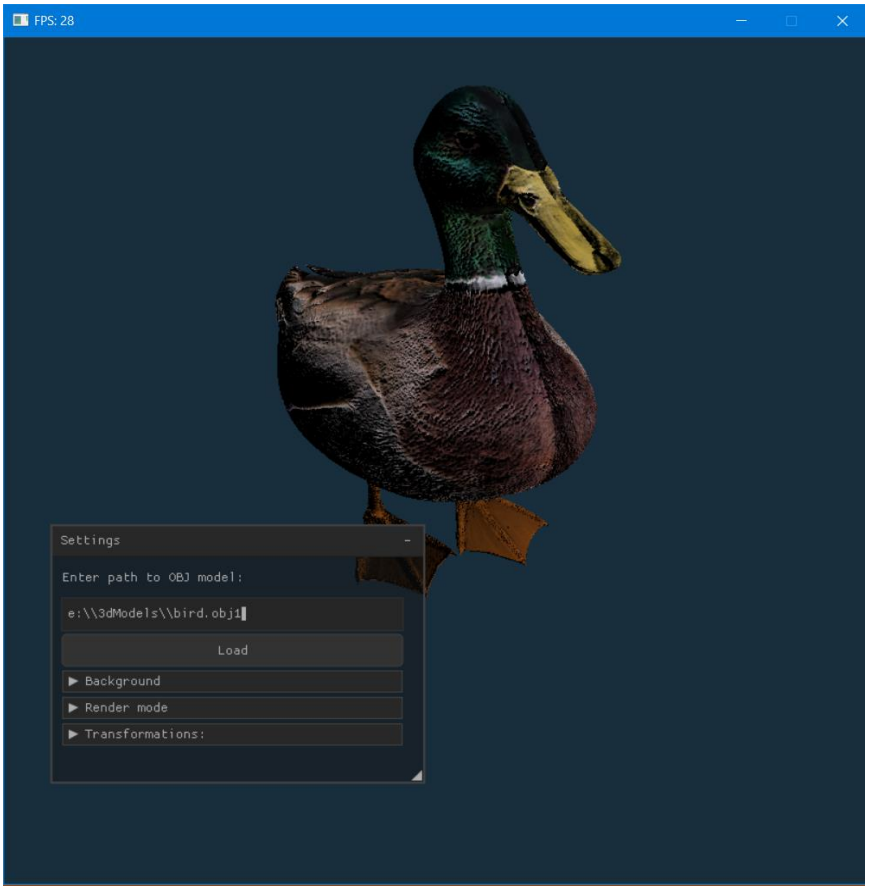


Рисунок 3.13 – Приклад завантаження довільного Wavefront OBJ файлу

Перше підменю під назвою “Background” відповідає за налаштування фону (рис. 3.14) тривимірного об’єкта. Воно містить елемент для зміни кольору фону (приклад показано на рис. 3.15). За його допомогою можна задати будь-який колір у зручному для

користувача режимі. Перемикач “Use TGA image as background” дозволяє задати будь-яке довільне зображення, задане у форматі TGA. При натисканні на елемент з’являється поле вводу шляху до зображення та кнопка “Set image”. Після натискання на кнопку фон змінюється на задане зображення (результат виконання показано на рис. 3.16).

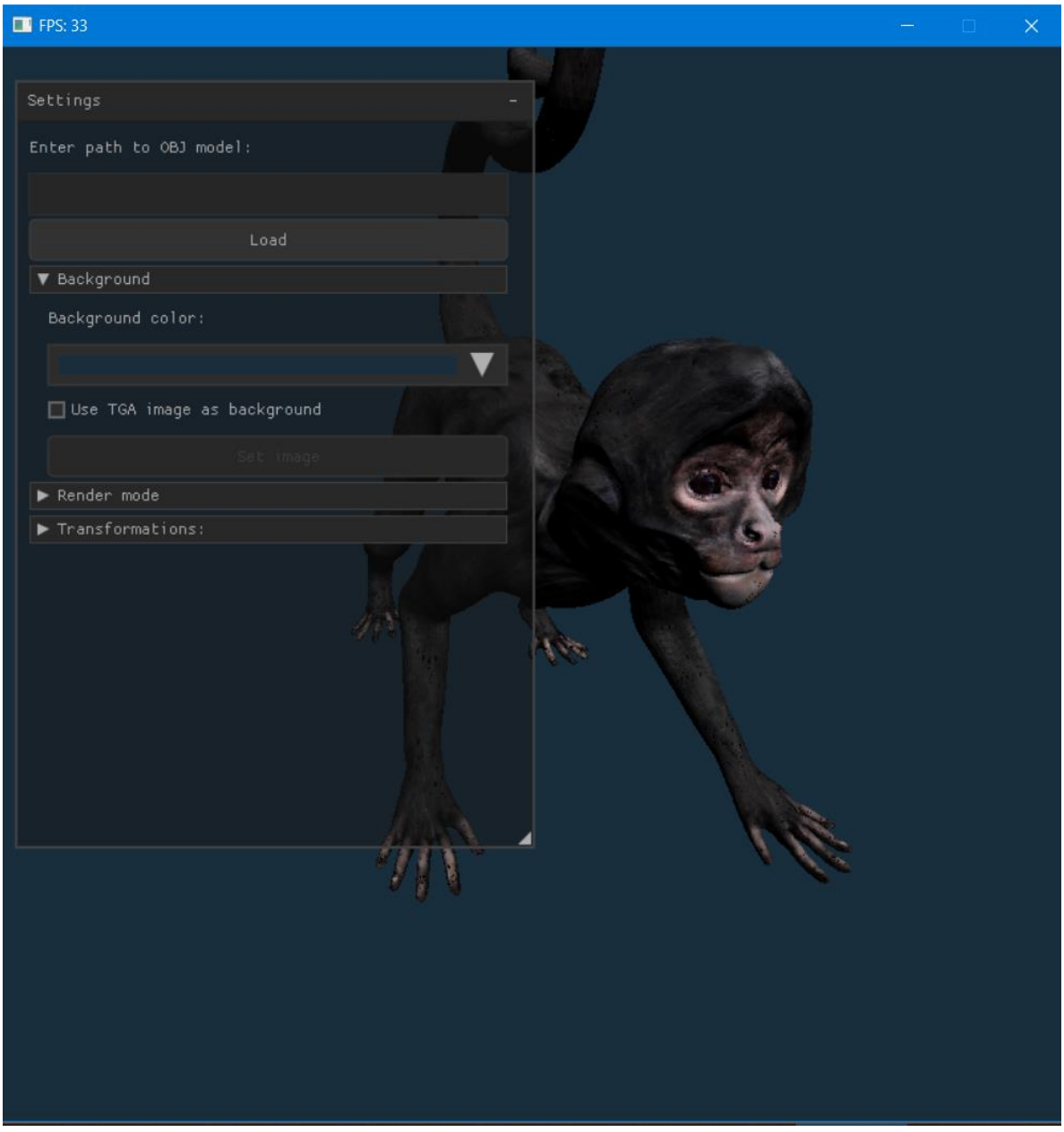


Рисунок 3.14 – Підменю “Background”

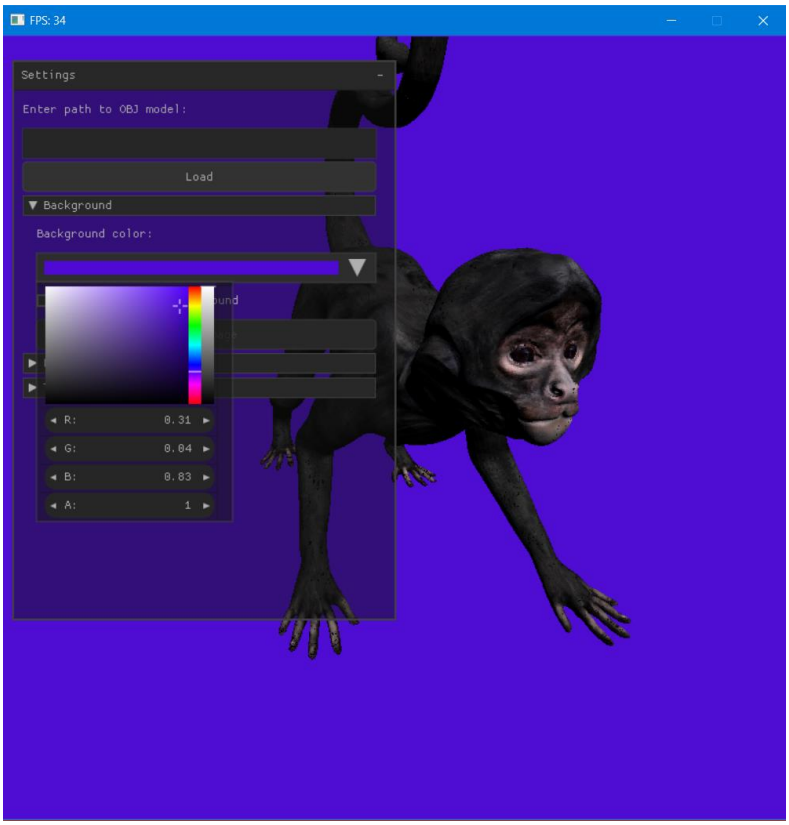


Рисунок 3.15 – Приклад зміни кольору фону

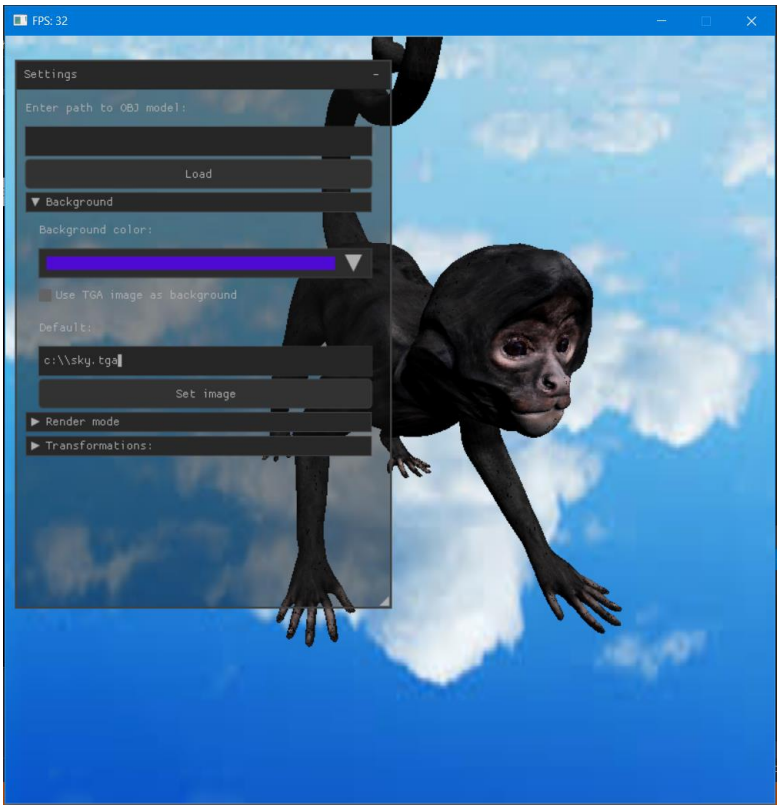


Рисунок 3.16 – Приклад використання зображення для фону

У підменю “Render mode” можна задати один із перелічених режимів візуалізації тривимірного об’єкта. За замовчуванням використовується режим “Shaders” (вигляд меню показано на рис. 3.17).

Розглянемо всі можливі режими:

- Wire – візуалізація за допомогою сітки трикутників (кожна сторона трикутника малюється з різним кольором для більшої наочності). Приклад використання режиму показано на рис. 3.18.
- Filled – візуалізація за допомогою трикутників із заливкою. Приклад використання режиму показано на рис. 3.19.
- Shaders – візуалізація за допомогою шейдерів. Приклад використання режиму показано на рис. 3.18.
- Shaders with wire – візуалізація за допомогою шейдерів, але з відображенням сітки трикутників. Приклад використання режиму показано на рис. 3.20.

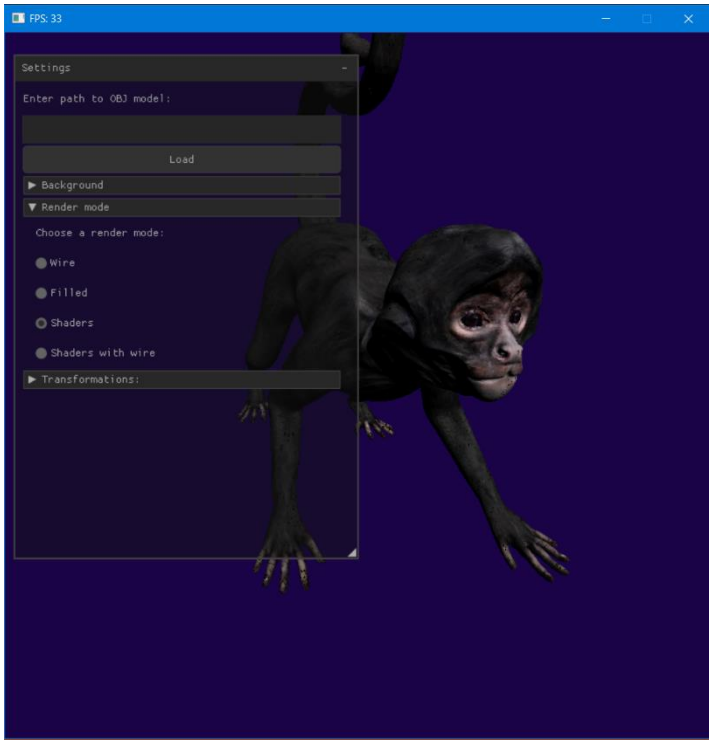


Рисунок 3.17 – Підменю режимів візуалізації (вибраний режим Shaders)

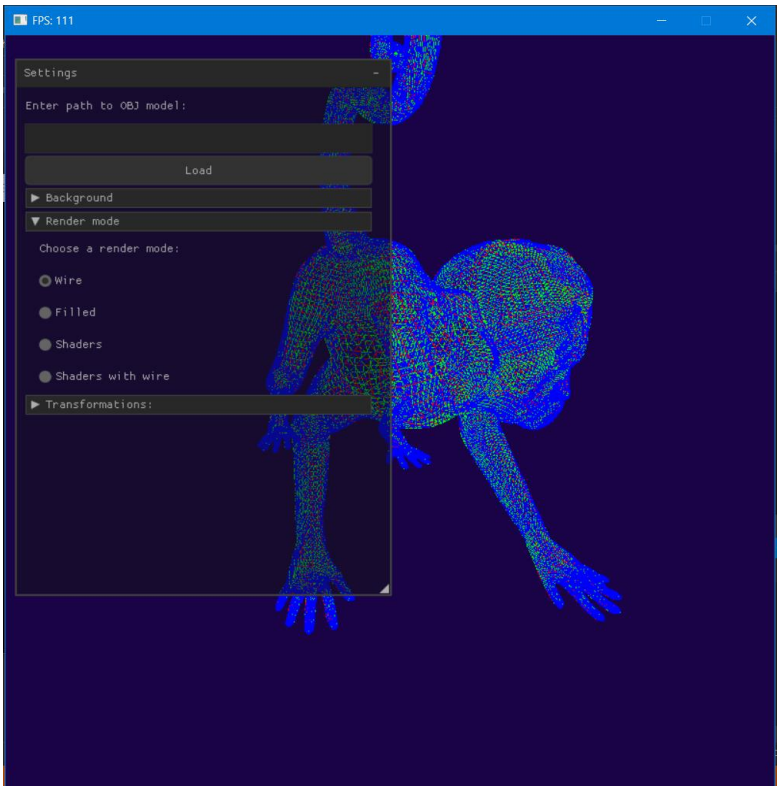


Рисунок 3.18 – Візуалізація в режимі Wire

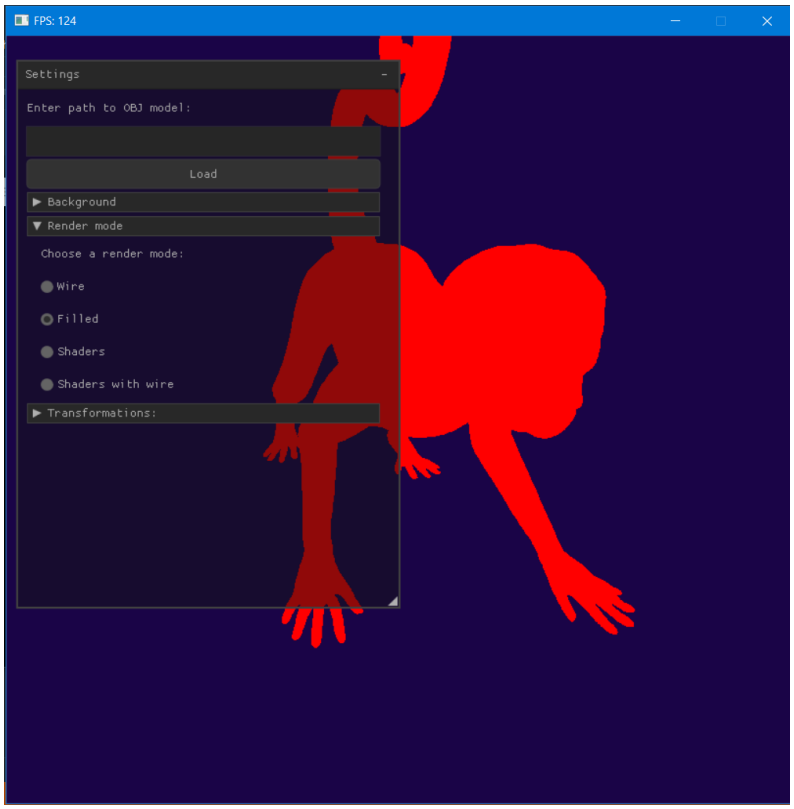


Рисунок 3.19 – Візуалізація в режимі Filled

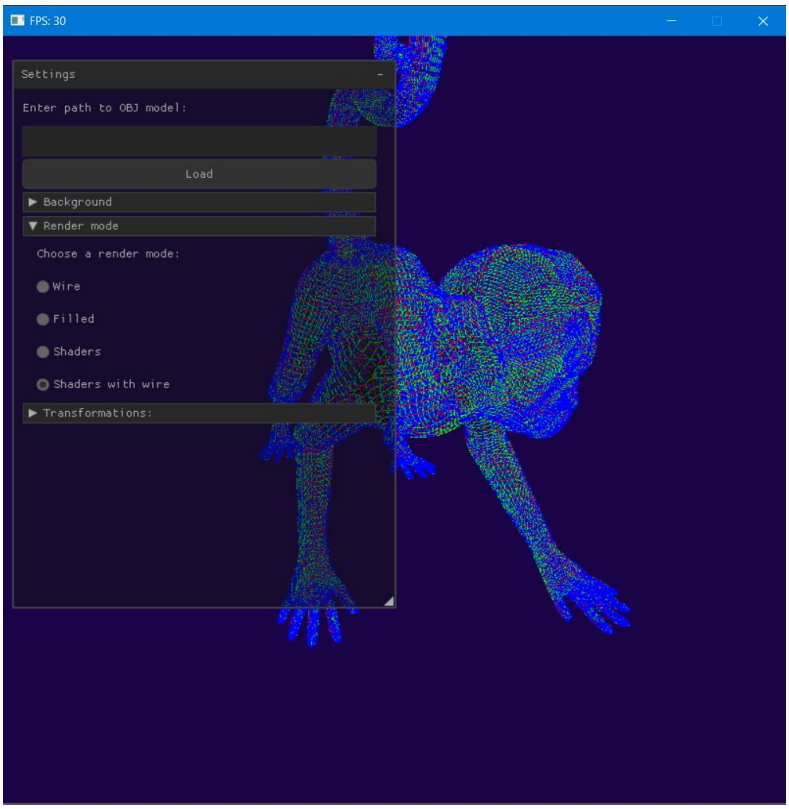


Рисунок 3.20 – Візуалізація в режимі Shaders with wire

Підменю “Transformation” містить елементи для пересування в тривимірному просторі. За його допомогою можна виконувати такі операції над зображенням як:

- Поворот об’єкта навколо заданої осі (підменю “Eye”). Приклад виконання операції показано на рис 3.21.
- Зміна положення джерела світла (підменю “Light direction”). Приклад виконання операції показано на рис 3.22.
- Зміна положення центру об’єкта (підменю “Center”). Приклад виконання операції показано на рис 3.23.
- Зміна вектору верху зображення, за його допомогою можна повертати об’єкт у довільному напрямку (підменю “Up vector”). Приклад виконання операції показано на рис 3.24.

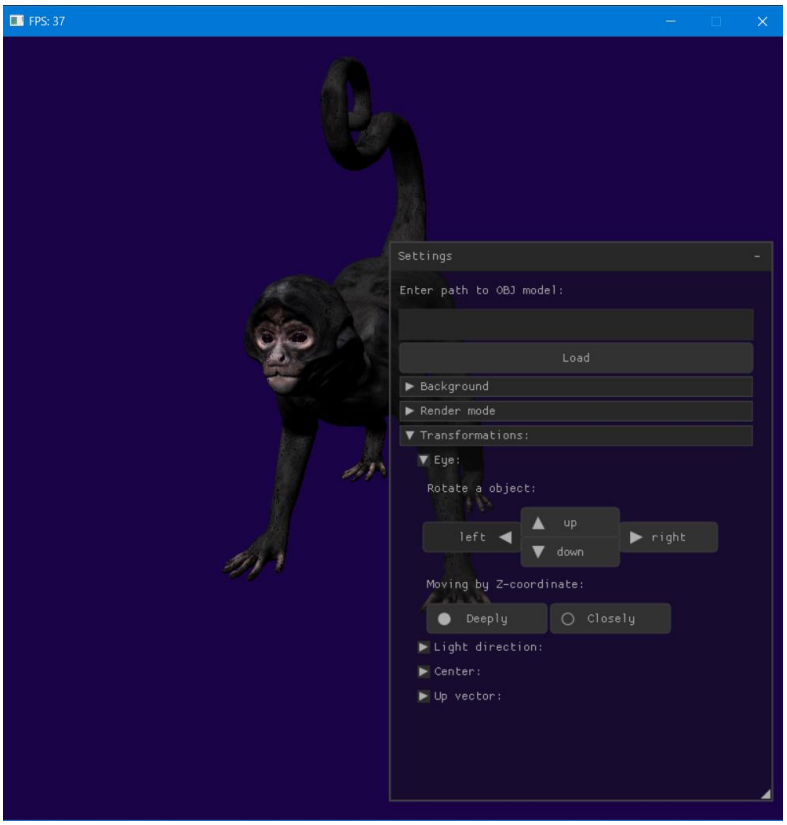


Рисунок 3.21 – Приклад повороту тривимірного об'єкта

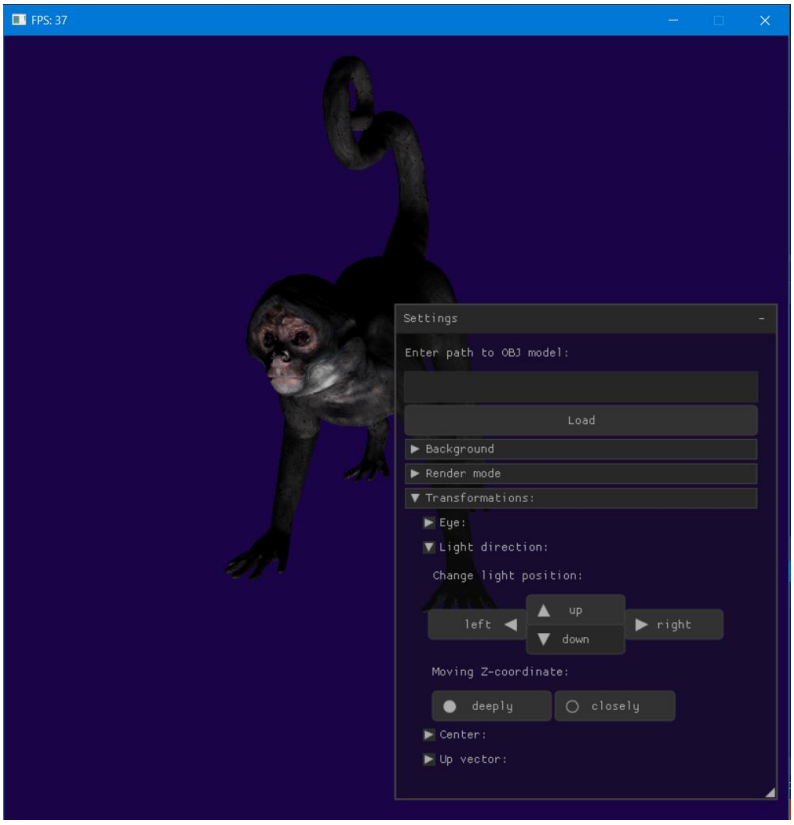


Рисунок 3.22 – Приклад зміни положення джерела світла

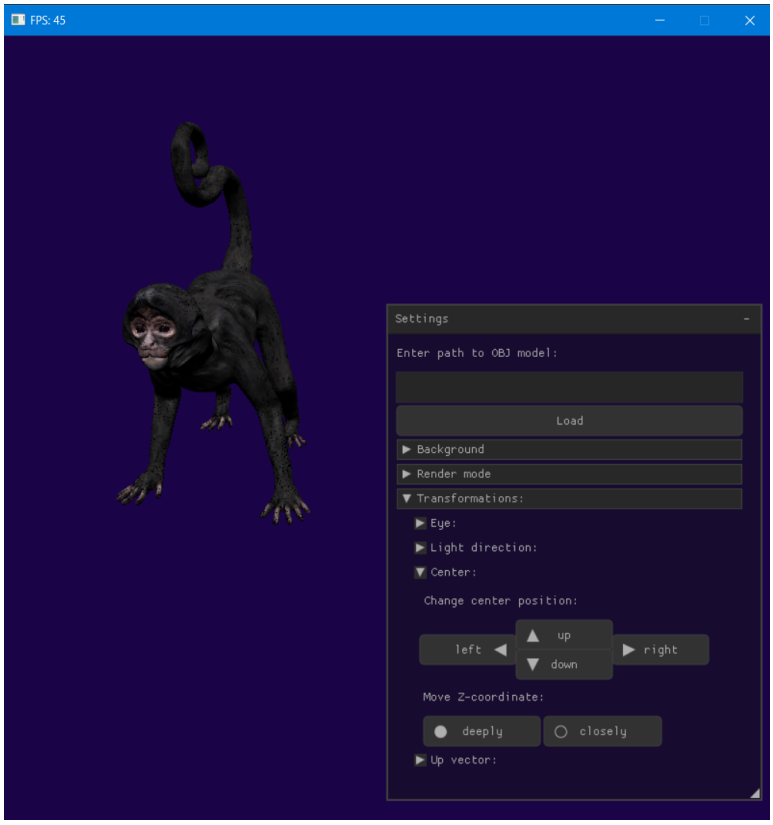


Рисунок 3.23 – Приклад зміни центру об'єкта

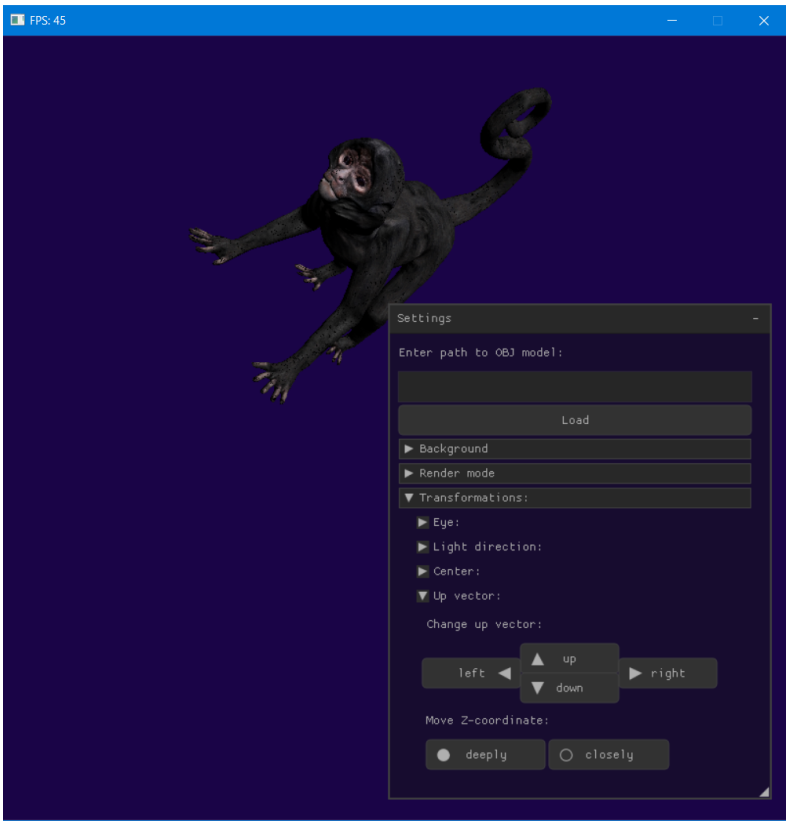


Рисунок 3.24 – Приклад зміни вектору верхньої точки об’єкта

4. ТЕСТУВАННЯ СИСТЕМИ

4.1. Способи тестування

Тестування програмної бібліотеки є важливим кроком у процесі розробки програмного забезпечення. Для тестування використано декілька методів:

- Функціональне тестування (functional testing) – методом «чорного ящику» (black box) – тестування на відповідність програмного продукту до ТЗ (перегляд структури та вихідного коду) та «білого ящику» (white box) – тестування на відповідність програмного продукту заданому функціоналу.
- Тестування швидкодії (performance testing) – для виявлення найбільшої обчислювальної спроможності графічної бібліотеки та загальної швидкодії.
- Тестування зручності та зрозумілості графічного інтерфейсу для звичайного користувача (usability testing) – досягнення максимальної зручності у роботі з бібліотекою та програмою для перегляду Wavefront OBJ зображень.

4.2. Аналіз швидкодії

Для відлагодження та оптимізації графічної бібліотеки використано спеціальну утиліту NVIDIA Visual Profiler – інструмент для профілювання продуктивності, який надає розробникам корисну інформацію для оптимізації CUDA C / C ++ додатків. Вперше представлений у 2008 році, Visual Profiler підтримує всі 350 мільйонів графічних процесорів NVIDIA з підтримкою CUDA, які поставляються з 2006 року на платформах Linux, Mac OS X і Windows. NVIDIA Visual Profiler доступний як частина інструментарію CUDA.

Як видно з рис. 4.1., на початку роботи програми незначний час займає виклик внутрішнього CUDA API (як показує профілізатор, час виконання займає 4532.481 мікросекунди). Далі значний проміжок часу займає виділення пам’яті на графічному пристрої для подальшого використання (284189.683 мікросекунди).

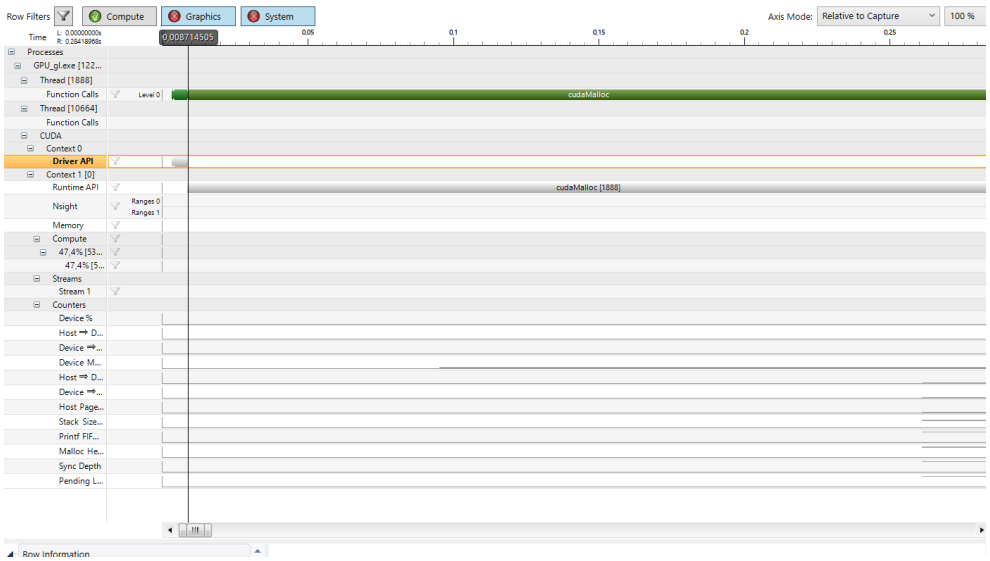


Рисунок 4.1 – Дослідження часу виділення пам’яті

Далі починається основний цикл програми. Як видно з рис. 4.2, для кожного прорахунку кадру за допомогою CUDA виконується спеціальна функція splitByThreads, яка розділяє виконання на безліч потоків, що виконуються паралельно. У кожному потоці запускається конвеєр візуалізації. Весь цей процес займає приблизно 32421 мікросекунд. Також перед викликом цієї функції та після її виконання здійснюється копіювання вмісту пам’яті в графічний прискорювач та з графічного прискорювача в пам’ять комп’ютера відповідно. Процес копіювання даних займає незначний час в порівнянні з обробкою даних на графічному прискорювачі та центральному процесорі.

Як видно на рис. 4.2, у проміжках часу, коли не виконується код графічного процесора, виконується код центрального процесора для підготовки наступного кадру. Цей процес займає приблизно 40% від загального часу.

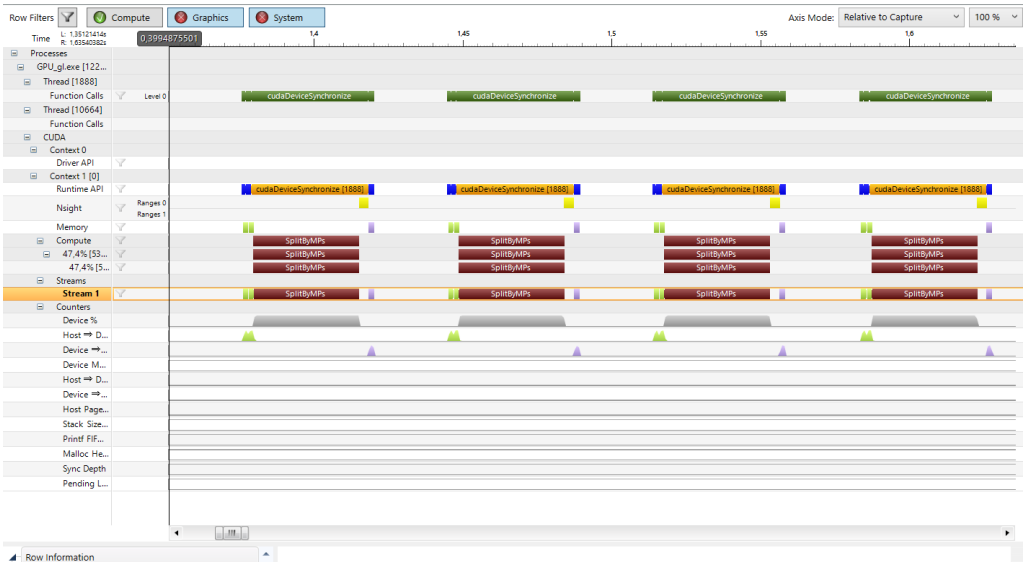


Рисунок 4.2 – Основний цикл бібліотеки

Можна зробити висновок, що процес візуалізації є оптимальним і в ньому відсутнє зайве виділення або копіювання пам'яті, що для гетерогенної архітектури є витратною за часом виконання операцією. Також, з рис. 4.2 видно, що для прискорення часу виконання є тільки два шляхи – збільшення швидкодії центрального або графічного процесору.

4.3. Порівняння швидкодії

Швидкодія бібліотеки безпосередньо залежить від потужності ПК, тому на кожному ПК швидкодія може значно відрізнятись. Також вона залежить від розміру та складності моделі, яка візуалізується. Чим більша кількість полігонів (трикутників) об'єкта, тим більше часу потрібно для його візуалізації. Не в меншій степені на швидкодію візуалізації впливає розмір текстур – чим більша текстура, тим більше часу потрібно на її обробку.

У тривимірній візуалізації швидкодію прийнято вимірювати у FPS (frames per second) – кількість кадрів в секунду на екрані комп'ютера. У професійних бібліотеках, для важких і навантажених сцен швидкодія звичайно становить приблизно 60-80 FPS. Для простих об'єктів FPS звичайно досягає 200-300.

У розробленій графічній бібліотеці в процесі візуалізації об'єктів середньої важкості досягається швидкодія порядку 33-50 FPS. Це пов'язано з використанням професійними бібліотеками графічних драйверів та розрахунків повністю на апаратному рівні за допомогою графічного прискорювача. Дана графічна бібліотека створена для надання

користувачу можливості відслідкувати кожний крок візуалізації тривимірного зображення. Саме тому всі функції візуалізації реалізовувались тільки за допомогою програмних засобів, що вплинуло на загальну швидкодію. Виведення згенерованого зображення на екран комп’ютера також забирає певний час. Оскільки сучасні ОС не дозволяють виводити зображення безпосередньо у відеопам’ять, зображення виводиться за допомогою сторонньої бібліотеки, яка є надлишковим рівнем абстракції та забирає близько 10-20% загального часу візуалізаці.

ВИСНОВКИ

Результатом виконання дипломного проекту є повністю реалізована графічна бібліотека, що дозволяє візуалізацію будь-якого тривимірного об’єкту в Wavefront OBJ форматі, накладання на заданий об’єкт довільної текстури, створення власних шейдерів для обробки об’єкту та відстеження кожного кроку виконання графічного конвеєру. Бібліотека підтримує базові операції перетворення, такі як поворот, масштабування тощо. На сьогодні подібних бібліотек, в яких можливо побачити кожний крок виконання, не існує.

Створена графічна бібліотека є хорошим навчальним ПЗ, яке дозволить прикладному програмісту зрозуміти та дослідити кожний крок візуалізації тривимірного зображення, легко модифікувати існуючий вихідний код, покращити засвоєння основних концепцій комп’ютерної візуалізації та ефективно їй використовувати при розробці графічних програм.

У проекті досліджено різні методи візуалізації зображення, математичні методи для побудови тривимірних зображень, алгоритми для покращення зображення, такі як Z-буферизація, тощо.

Розроблено зручне API для відлагодження бібліотеки. За допомогою CUDA SDK реалізовано гетерогенну архітектуру, яка дозволяє перенести важкі обчислення з центрального процесора на графічний прискорювач, який виконує операції над даними за допомогою великої кількості апаратних ядер графічного прискорювача. Таким чином, всі витратні розрахунки виконуються на графічному прискорювачі. Вивільняється час центрального процесора, що дозволяє йому виконувати обчислення для підготовки вже наступного кадру. Така реалізація забезпечує швидку візуалізацію в реальному часі.

Створенні програмні засоби повністю задовольняють вимогам технічного завдання на дипломне проектування.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Edward A.D., Dave S.H. Interactive Computer Graphics: A Top-Down Approach With Shader-Based Opengl, Pearson; 6 edition, 2011.

2. Роберт Сикорд. Безопасное программирование на C и C++, Издательство Вильямс, 2-е издание, 2016.

3. Пономаренко С.И. Пиксел и вектор. Принципы цифровой графики, BHV-СПб, 2002.

4. V. Scott Gordon, John Clevenger. COMPUTER GRAPHICS PROGRAMMING IN OPENGL WITH C++, Mercury Learning and Information, 2018.

5. Kyle H.P. Practical Shader Development: Vertex and Fragment Shaders for Game Developers, Apress, 1st ed. Edition, 2019.

6. Georg Glaeser , Hans-Peter Schröcker. Handbook of Geometric Programming Using Open Geometry GL (Springer Professional Computing), Springer, 2002.

7. Eric L.C. Mathematics for 3D Game Programming and Computer Graphics, Cengage Learning PTR; 3rd edition, 2011.

8. Siddharth S.A. C++ Game Development By Example: Learn to build games and graphics with SFML, OpenGL, and Vulkan using C++ programming, Packt Publishing, 2019.

9. Jonathan M. D. Hill. An introduction to the data-parallel paradigm. Department of Computer Science, 1994.

10. Fletcher D. 3D Math Primer For Graphics and Game Development. Jones & Bartlett Learning, 1 edition, 2002.

11. Dr. Jeremy Kun. A Programmer's Introduction to Mathematics. CreateSpace Independent Publishing Platform, 1 edition, 2018